Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Computer Communications 26 (2003) 1484–1503

computer communications

www.elsevier.com/locate/comcom

Distributed dynamic capacity contracting: an overlay congestion pricing framework

Murat Yuksel*, Shivkumar Kalyanaraman

ECSE Department, Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180, USA

Received 18 December 2002; accepted 18 December 2002

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 differentiated-services (diff-serv) domain. 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. We develop a pricing scheme within the Distributed-DCC framework and investigate several issues such as optimality of prices, fairness of rate allocation.

We also introduce two pricing architectures 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. We, then, adapt Distributed-DCC framework to these architectures, and evaluate the two architectures by extensive simulation.

Keywords: Congestion pricing; Differentiated-services; Distributed Dynamic Capacity Contracting

1. Introduction

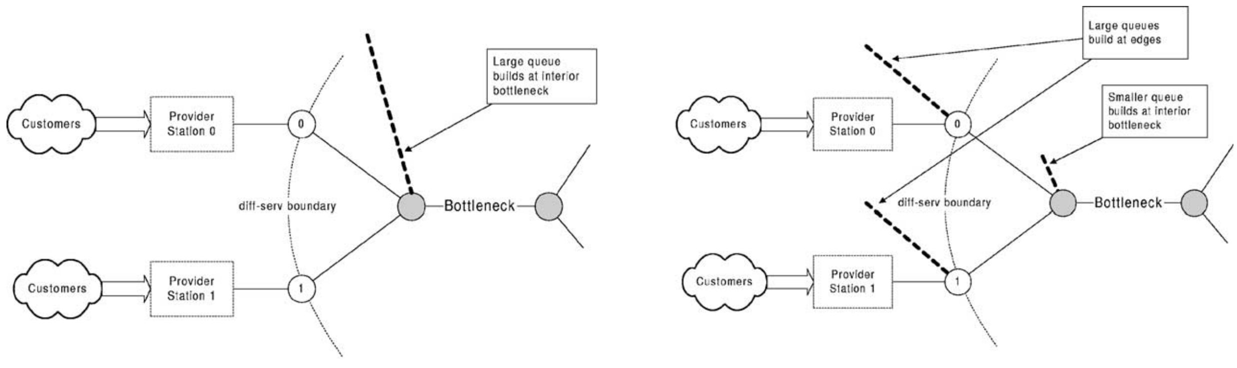
Implementation of congestion pricing still remains a challenge, although several proposals have been made, e.g. Refs. [1–3]. Among many others, two major implementation obstacles can be defined: need for *timely feedback* to users about the price, determination of *congestion information* in an efficient, low-overhead manner.

The first problem, timely feedback, is relatively very hard to achieve in a wide area network such as the Internet. In Ref. [4], the authors showed that users do want 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 by 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 by pricing, service prices must be updated very frequently (i.e. 2–3 s since RTT is expressed in terms of milliseconds for most cases in the Internet). In order to solve this time-scale problem for dynamic pricing, we propose two solutions, which lead to two different pricing 'architectures':

- *By placing intelligent intermediaries* (*i.e. software or hardware agents*) *between users and the provider.* This way it is possible for the provider to update prices frequently at low time-scales, since price negotiations will be made with a software/hardware agent rather than a human. Since the provider will not employ any congestion control mechanism for its network and try to control congestion by only pricing, we call this pricing architecture as *Pricing for Congestion Control* (*PFCC*).
- *By overlaying pricing on top of an underlying congestion control mechanism.* This way it is possible to enforce tight control on congestion at small time-scale, while performing pricing at time-scales large enough for human involvement. The provider implements a congestion control mechanism to manage congestion in its network. So, we call pricing architecture as *Pricing over Congestion Control* (*POCC*).

* Corresponding author.

0140-3664/03/$ - see front matter © 2003 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0140-3664(03)00033-1



(a) Pricing for Congestion Control (PFCC)

(a) Pricing over Congestion Control (POCC)

Fig. 1. Different pricing architectures with/without edge-to-edge congestion control.

Big-picture of the two pricing architectures PFCC and POCC are shown in Fig. 1. We will describe PFCC and POCC later in Section 3.

The second problem, congestion information, is also very hard to solve in a way that does not require 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 framework 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. 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 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):

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

Fig. 2 shows the big picture of DCC framework. Customers can only access network core by making contracts with the provider stations placed at the edge routers. The stations offer contracts (i.e. V_{max} and T) to fellow users. Access to these available contracts can be done in different ways, what we call *edge strategy*. Two basic edge strategies are 'bidding' (many users bids for an available contract) or 'contracting' (users negotiate P_v with the provider for an available contract).

Notice that, in DCC framework, provider stations can implement dynamic pricing schemes. Particularly, they can implement congestion-based pricing schemes, if they have actual information about congestion in 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.

However, in DCC, we assumed that all the provider stations advertise the same price value for the contracts, which is very costly to implement over a wide area network. This is simply because the price value cannot be communicated to all stations at the beginning of each contract. In this paper, we relax this assumption by letting the stations to calculate the prices locally and advertise different prices than the other stations. We call this new version of DCC as *Distributed-DCC*. We introduce ways of managing the overall coordination of the stations.

As a fundamental difference between Distributed-DCC and the well-known dynamic pricing proposals (e.g. Kelly et al.'s proposal [8], Low et al.'s proposal [9]) in the area lies in the manner of price calculation.

In Distributed-DCC, the prices are calculated on an edge-to-edge basis, while traditionally it has been proposed that prices are calculated at each local link and fed back to users. To make it more concrete, Fig. 3a and b shows the case of

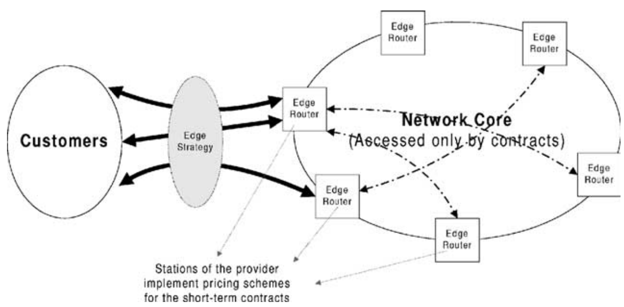
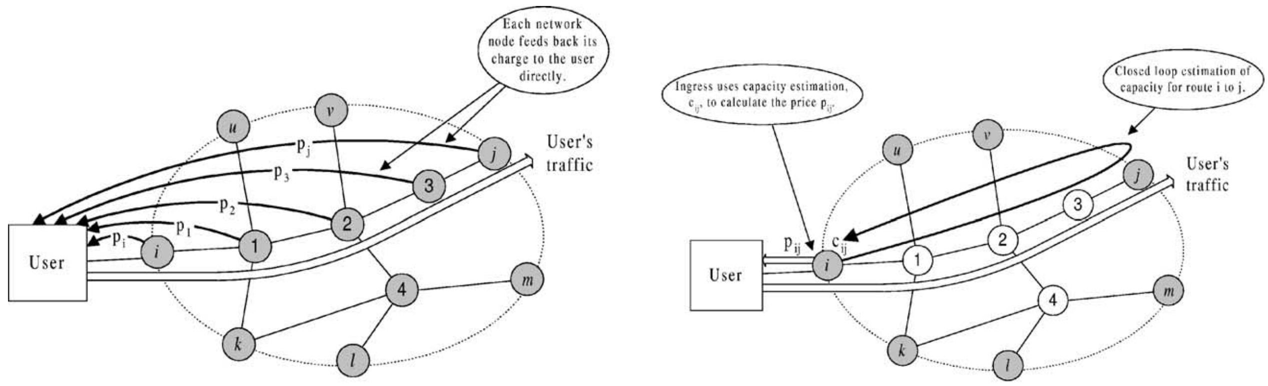


Fig. 2. DCC framework on diff-serv architecture.



(a) Low et al.' pricing framework. (b) Distributed-DCC framework.

Fig. 3. Components of Distributed-DCC with Low et al.'s pricing framework in terms of price calculation.

Distributed-DCC and the case of Low et al.'s framework. Gray nodes are the ones that participate in price calculation for a user. In Distributed-DCC, basically, the links on a flow's route are abstracted out by edge-to-edge capacity estimation and the ingress node communicates with the corresponding egress node to observe congestion on the route. Then, the ingress node uses the estimated capacity and the observed congestion information in price calculation. However, in Low et al.'s framework, each link calculates its own price and sends it to the user, and the user pays the aggregate price. So, Distributed-DCC is better in terms of implementation requirements, while Low et al.'s framework is better in terms of optimality. Distributed-DCC trades off some optimality in order to enable implementation of dynamic pricing. Amount of lost optimality depends on the closed-loop edge-to-edge capacity estimation.

The paper is organized as follows. In Section 3, we position our work and briefly survey relevant work in the area. In Section 3, we present PFCC and POCC pricing architectures motivated by the time-scale issues mentioned above. In Section 6 we describe properties of Distributed-DCC framework according to the PFCC architecture. Then, in Section 7, we revise Distributed-DCC's definition in Section 6 and adapt it to the POCC architecture. In other words, we mainly define the Distributed-DCC framework in Section 6, and then in Section 6 we add necessary components to Distributed-DCC in order to adapt it to POCC. Next in Section 5, we define a pricing scheme Edge-to-Edge Pricing (EEP) which can be implemented in the defined Distributed-DCC framework. We study optimality of EEP for different forms of user utility functions and consider effect of different parameters such as user's budget, user's elasticity. In Section 8, according to the descriptions of Distributed-DCC framework and EEP scheme, we simulate Distributed-DCC in the two architectures PFCC and POCC. With the simulation results, we compare Distributed-DCC's performance in PFCC and POCC architectures. We finalize with summary and discussions in Section 9.

2. 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 *a priori* to service vs. *a posteriori* to service.

Although there are opponents to dynamic pricing in the area [10–12], 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 [13], Kelly et al.'s Proportional Fair Pricing (PFP) [8], Semret et al.'s Market Pricing [3,14], and Wang and Schulzrinne's Resource Negotiation and Pricing (RNAP) [2,15]. Odlyzko's Paris Metro Pricing (PMP) [16] is an example of static pricing proposal. Clark's Expected Capacity [17,18] and Cocchi et al.'s Edge Pricing [19] 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 Ref. [20], 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.

In the area, another proposal that mainly focused on implementation issues of congestion pricing on diff-serv is RNAP [2,15]. 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 proposals 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.

3. Pricing architectures: PFCC vs. POCC

In this section, we introduce two new pricing architectures that are mainly motivated by time-scale problems regarding control of congestion by pricing (details in Section 1).

3.1. Pricing for Congestion Control

In this pricing architecture, provider attempts to solve congestion problem of its network just by congestion pricing. In other words, the provider tries to control congestion of its network by changing service prices. The problem here is that the provider will have to change the price very frequently such that human involvement into the price negotiations will not be possible. This problem can be solved by running intermediate software (or hardware) agents between end-users and the provider. The intermediate agent receives inputs from the end-user at large time-scales, and keeps negotiating with the provider at small time-scales. So, intermediate agents in PFCC architecture are very crucial in terms of acceptability by users.

If PFCC architecture is not employed (i.e. providers do not bother to employ congestion pricing), then congestion control will be left to the end-user as it is in the current Internet. Currently in the Internet, congestion control is totally left to end-users, and common way of controlling congestion is TCP and its variants. However, this situation leave open doors to non-cooperative users who do not employ congestion control algorithms or at least employ congestion control algorithms that violates fairness objectives. For example, by simple tricks, it is possible to make TCP connection to capture more of the available capacity than the other TCP connections.

The major problem with PFCC is that development of user-friendly intermediate agents is heavily dependent on user opinion, and hence requires significant amount of research. A study of determining user opinions is available in Ref. [4]. In this paper, we do not focus development of intermediate agents.

3.2. Pricing over Congestion Control

Another way of approaching the congestion control problem by pricing is to overlay pricing on top of congestion control. This means the provider undertakes the congestion control problem by itself, and employs an underlying congestion control mechanism for its network. This way it is possible to enforce tight control on congestion at small time-scales, while maintaining human involvement into the price negotiations at large time-scales. Fig. 1 shows the difference between POCC (with congestion control) and PFCC (without congestion control) architectures.

So, assuming that there is an underlying congestion control scheme, the provider 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 congestion control scheme accordingly. This way, it will be possible to favor some traffic flows with higher willingness-to-pay (i.e. budget) 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 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.

Since our main focus is to implement pricing in 'diff-serv environment', we assume that the provider employs 'edge-to-edge' congestion control mechanisms under the pricing protocol on top. So, in diff-serv environment, overlaying pricing on top of edge-to-edge congestion control raises two major problems:

1. *Parameter mapping*. Since the pricing protocol wants to allocate network capacity according to the user incentives (i.e. the users with greater budget should get more capacity) that changes dynamically over time, it is a required ability set corresponding parameters of the underlying edge-to-edge congestion control mechanism such that it allocates the capacity to the user flows according to their incentives. So, this raises need for a method of mapping parameters of the pricing scheme to the parameters of the underlying congestion control mechanism. Notice that this type of

mapping requires the edge-to-edge congestion control mechanism to be able to provide parameters that tunes the rate being given to edge-to-edge flows.

2. *Edge queues*. The underlying edge-to-edge congestion control scheme will not always allow all the traffic admitted by the pricing protocol, which will cause queues to build up at network edges. So, management of these edge queues is necessary in POCC architecture. Fig. 1a and b compare the situation of the edge queues in the two cases when there is an underlying edge-to-edge congestion control scheme and when there is not.

Another problem is that the overall performance of the system will be dependent on not only the pricing protocol's performance, but also the performance of the underlying congestion control scheme. For instance, if the underlying congestion control scheme does not allow the network to be utilized more than 80% for some internal reason, then the utilization provided by the overall system will be limited by 80%.

4. Distributed-DCC framework

Distributed-DCC framework is specifically designed for diff-serv environment, 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. We can summarize essence of Distributed-DCC in two items:

- Since upgrade to all routers is not possible to implement, pricing should happen on an *edge-to-edge* basis which only requires upgrades to edge routers.
- Provider should employ short-term contracts in order to have ability to change prices frequently enough such that congestion-pricing can be enabled.

Distributed-DCC framework has three major components as shown in Fig. 4: *Logical Pricing Server* (*LPS*), *Ingress Stations*, and *Egress Stations*. Solid lined arrows in the figure represent control information being transmitted among the components. Basically, Ingress stations negotiate with customers, observe customer's traffic, and make estimations about customer's demand. Ingress stations inform corresponding Egress stations about the observations and estimations about each edge-to-edge flow.

Egress stations detect congestion by monitoring edge-to-edge traffic flows. Based on congestion detections, Egress stations estimate available capacity for each edge-to-edge flow, and inform LPS about these estimations.

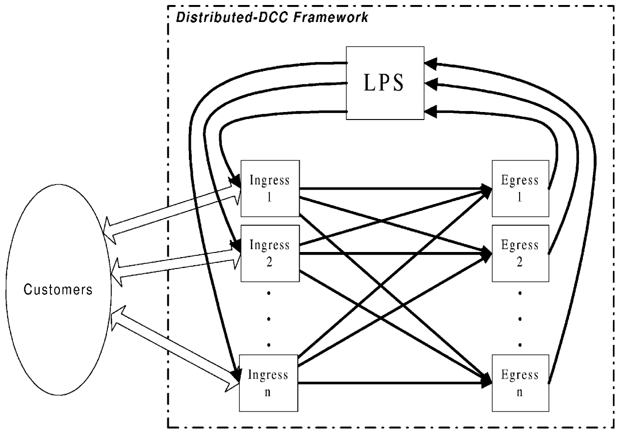


Fig. 4. Components of Distributed-DCC framework. Solid lined arrows represent flow of control information necessary for price calculation. In PFCC architecture, communication with LPS must be at very short time-scales (i.e. each short-term contract). However, in POCC, LPS is accessed at longer time-scales (i.e. parameter remapping instants).

LPS receives capacity estimations from Egress stations, and allocates the network available capacity to edge-to-edge flows according to different criteria (such as fairness, price optimality).

Below, we describe functions and sub-components of these three components in detail. Also, to ease understanding of the framework, we show important parameters, their symbols and their descriptions in Table 1.

4.1. Ingress station i

Fig. 5 shows sub-components of Ingress station i in the framework. Ingress i includes two sub-components: *Pricing Scheme* and *Budget Estimator*.

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 using flow i to j is charged the price p_{ij}. Pricing Scheme is the sub-component that calculates price p_{ij} for each edge-to-edge flow starting at Ingress i. It uses allowed flow capacities c_{ij} and other local information (such as $\hat{b}_{ij}$), in order to calculate price p_{ij}. The station, then, uses p_{ij} in negotiations with customers. We will describe a simple pricing scheme EEP later in Section 5. However, it is possible to implement several other pricing schemes by using the information available at Ingress i. Other than EEP, we implemented another pricing scheme, Price Discovery, which is available in Ref. [21].

Also, the ingress i uses the total estimated network capacity C in calculating the $V_{\max}$ contract parameter defined in Eq. (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.

Budget Estimator is the sub-component that observes demand for each edge-to-edge flow. We implicitly assume

Table 1
List of parameters in Distributed-DCC framework

Parameter	Symbol	Description
Contract length (s)	T	Length of contracts
Observation interval (s)	O	Time-scale of observations at Egress
LPS interval (s)	L	Time-scale of communication between LPS and provider stations
Edge-to-edge price (\$/Mb)	p_{ij}	Unit price for traffic flow from i to j
Budget estimation (\$)	$\hat{b}_{ij}$	Estimation for budget of flow from i to j
Updated budget estimation (\$)	b_{ij}	Budget estimation for flow from i to j adjusted by Fairness Tuner
Estimated network capacity (Mb/s)	C	Estimation for total network capacity
Estimated capacity (Mb/s)	$\hat{c}_{ij}$	Estimation of available capacity for flow i to j
Allowed capacity (Mb/s)	c_{ij}	Capacity given by Capacity Allocator to flow i to j
Flow input rate at ingress (Mb/s)	x_{ij}	Arrival rate of flow i to j at ingress i
Flow output rate at egress (Mb/s)	μ_{ij}	Departing rate of flow i to j at egress j
Estimated flow cost	$\hat{r}_{ij}$	Estimation for cost incurred by flow i to j
Fairness coefficient	α	Tuner for fairness type of Fairness Tuner

that user's 'budget' represents user's demand (i.e. willingness-to-pay). So, Budget Estimator estimates budget $\hat{b}_{ij}$ of each edge-to-edge traffic flow.[1] We will describe a simple algorithm that calculates $\hat{b}_{ij}$ later in Section 4.4.1.

4.2. Egress station j

Fig. 6 shows sub-components of Egress station j in the framework: *Congestion Detector*, *Congestion-Based Capacity Estimator*, *Flow Cost Analyzer*, and *Fairness Tuner*.

Congestion Detector implements an algorithm to detect congestion in network core by observing traffic arriving at Egress j. Congestion detection can be done in several ways. We assume that interior routers mark (i.e. sets the ECN bit) the data packets if their local queue exceeds a threshold. Congestion Detector generates a 'congestion indication' if it observes a marked packet in the arriving traffic.

Congestion-Based Capacity Estimator estimates available capacity $\hat{c}_{ij}$ for each edge-to-edge flow exiting at Egress j. In order to calculate $\hat{c}_{ij}$, it uses congestion indications from Congestion Detector and actual output rates μ_{ij} of the flows. The crucial property of Congestion-Based Capacity Estimator is that, it estimates capacity in a congestion-based manner, i.e. it decreases the capacity estimation when there is congestion indication and increases when there is no congestion indication. This makes the prices *congestion-sensitive*, since Pricing Scheme at Ingress calculates prices based on the estimated capacity. An example algorithm for Congestion-Based Capacity Estimator will be described later in Section 4.4.2.

Flow Cost Analyzer determines cost of each traffic flow (e.g. number of links traversed by the flow, number of bottlenecks traversed by the flow, amount of queuing delay caused by the flow) exiting at Egress j. Cost incurred by each flow can be several things: number of traversed links, number of traversed bottlenecks, amount of queuing delay caused. We assume that number of bottlenecks is a good representation of the cost incurred by a flow. In Appendix A, we define an algorithm ARBE, which estimates number of bottleneck traversed by a flow. ARBE outputs estimated number of bottlenecks $\hat{r}_{ij}$ traversed by the flow from ingress i to egress j.

LPS, as will be described in Section 4.3, allocates capacity to edge-to-edge flows based on their budgets. The flows with higher budgets are given more capacity than the others. So, Egress j can penalize/favor a flow by increasing/decreasing its budget $\hat{b}_{ij}$. Fairness Tuner is the component that updates $\hat{b}_{ij}$. So, Fairness Tuner penalizes or favors the flow from ingress i by updating its estimated budget value, i.e. $b_{ij} = f(\hat{b}_{ij}, \hat{r}_{ij}, \langle parameters \rangle)$ where $\langle parameters \rangle$ are other optional parameters that may be used for deciding how much to penalize or favor the flow. For example, if the flow ingress i is passing through more congested areas than the other flows, Fairness Tuner can penalize this flow by reducing its budget estimation $\hat{b}_{ij}$. We will describe an algorithm for Fairness Tuner later in Section 4.4.4.

Egress j sends $\hat{c}_{ij}$s (calculated by Congestion-Based Capacity Estimator) and b_{ij}s (calculated by Fairness Tuner) to LPS.

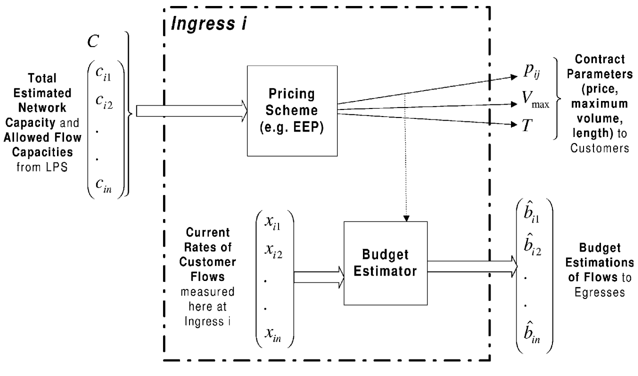


Fig. 5. Major functions of ingress i.

[1] Note that edge-to-edge flow does not mean an individual user's flow. Rather it is the traffic flow that is composed of aggregation of all traffic going from one edge node to another edge node.

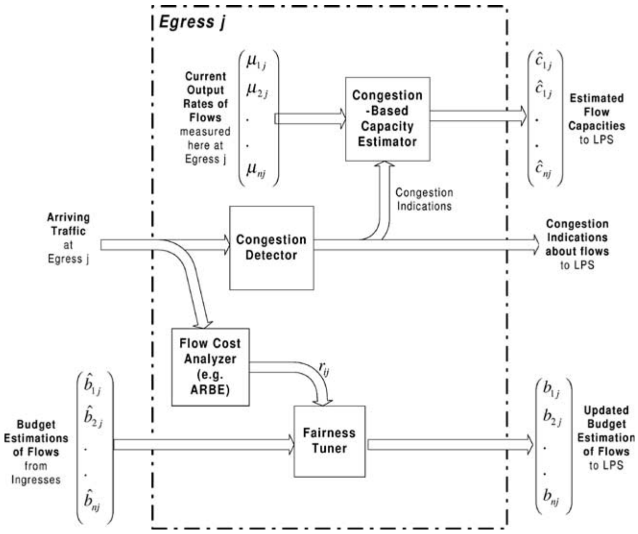


Fig. 6. Major functions of egress j.

4.3. Logical Pricing Server

Fig. 7 shows basic functions of LPS in the framework. LPS receives information from egresses and calculates *allowed capacity* c_{ij} for each edge-to-edge flow. The communication between LPS and the stations take place at every *LPS interval L*. There is only one major sub-component in LPS: Capacity Allocator.

Capacity Allocator receives $\hat{c}_{ij}$s, *bij*s and congestion indications from Egress Stations. It calculates allowed capacity c_{ij} for each flow. Calculation of c_{ij} values is a complicated task which depends on internal topology. In general, the flows should share capacity of the same bottleneck in proportion to their budgets. We will later define a generic algorithm ETICA for Capacity Allocator in Section 4.4.3.

Other than functions of Capacity Allocator, LPS also calculates total available network capacity C, which is necessary for determining the contract parameter $V_{\max}$ at Ingresses. LPS simply sums $\hat{c}_{ij}$ to calculate C.

LPS can be implemented in a centralized or distributed manner (see Section 6.1).

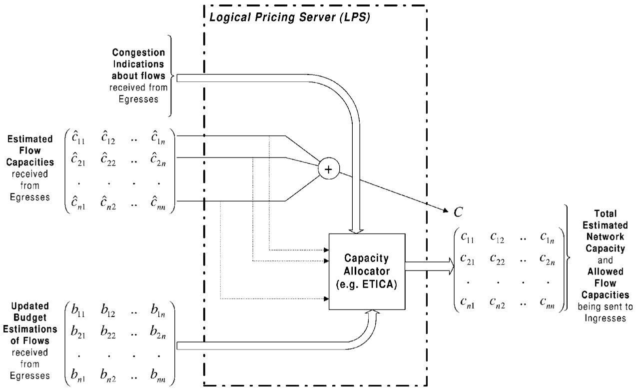


Fig. 7. Major functions of LPS.

4.4. Sub-components

4.4.1. Budget Estimator

At Ingress i, Budget Estimator performs a very trivial operation to estimate budgets $\hat{b}_{ij}$ of each flow starting at Ingress i. 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 Ingress i 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.

4.4.2. Congestion-based Capacity Estimator

The essence of Congestion-Based Capacity Estimator is to decrease the capacity estimation when there is congestion indication(s) and to increase it when there is no congestion indication. In this sense, several capacity estimation algorithms can be used, e.g. Additive Increase Additive Decrease (AIAD), Additive Increase Multiplicative Decrease (AIMD). We now provide a full description of such an algorithm.

At Egress j, given congestion indications from Congestion Detector and output rate μ_{ij} of flows, Congestion-Based Capacity Estimator implements the following algorithm for each flow from Ingress i : Let O be *observation intervals* at which the estimator makes an observation about congestion status of the network. The estimator identifies each observation interval as *congested* or *non-congested*. Basically, an observation interval is congested if a congestion indication was received from Congestion Detector during that observation interval. At the end of each observation interval t, the estimator updates the estimated capacity $\hat{c}_{ij}$ as follows:

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

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

4.4.3. ETICA: Edge-to-edge, Topology-Independent Capacity Allocation

Firstly, note that LPS is going to implement ETICA algorithm as a Capacity Allocator (see Fig. 7). So, we will refer to LPS throughout the description of ETICA below.

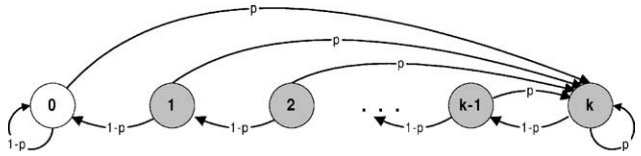


Fig. 8. States of an edge-to-edge flow in ETICA algorithm. The states $i > 0$ are congested states and the state $i = 0$ is the non-congested state, represented with gray and white colors, respectively.

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).

Again at LPS, let K_{ij} determine the state of f_{ij}. If $K_{ij} > 0$, LPS determines f_{ij} as congested. If not, it determines f_{ij} as non-congested. At every LPS interval t, LPS calculates K_{ij} as follows:

$$K_{ij}(t) = \begin{cases} \hat{k}, & \text{congestion in } t-1 \\ K_{ij}(t-1) - 1, & \text{no congestion in } t-1 \end{cases} \quad (2)$$

where $\hat{k}$ is a positive integer. Notice that $\hat{k}$ parameter defines how long a flow will stay in 'congested' state after the last congestion indication. So, in other words, $\hat{k}$ defines the time-line to determine if a congestion indication is recent or not. According to these considerations in ETICA algorithm, Fig. 8 shows states of an edge-to-edge flow given that probability of receiving a congestion indication in the last LPS interval is p. Gray states are the states in which the flow is congested, and the single white state is the non-congested state. Observe that number of congested states (i.e. gray states) is equal to $\hat{k}$ which defines to what extent a congestion indication is recent.[2]

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

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

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 sub-grouping the congested flows that are passing through the same bottleneck.

In short, the ETICA algorithm basically says that a flow in one of its congested states gets a share[3] of the total capacity of the congested flows (i.e. C_c). If the flow is in its in 'non-congested' state, then it uses its own capacity.

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. Dynamics of the algorithm will be understood more clearly after the simulation experiments in Section 8.

4.4.4. Fairness Tuner

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

- *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 [22], proportional fairness [8]. 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. 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.

For a better understanding of proportional fairness and max-min fairness, we study them in terms of social welfare maximization with a canonical example in Appendix B.

To achieve the fairness objectives defined in the above itemized list, we introduce new parameters for tuning rate allocation to flows. In order to penalize flow i to j, the egress

[2] Note that instead of setting K_{ij} to $\hat{k}$ at every congestion indication, more accurate methods can be used in order to represent self-similar behavior of congestion epochs. For simplicity, we proceed with the method in Eq. (2).

[3] Note that in this definition of ETICA, we defined this 'share' as the ratio of b_{ij}/B_c which is based on f_{ij}'s monetary value with respect to monetary value of all congested flows F_c. This is because our main goal is to 'price' effectively. However, one can define this share according to other criteria (such as equal to all congested flows), which makes it possible to use ETICA for completely rate allocation purposes.

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, Fairness Tuner 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 move 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. ARBE, in Appendix A, calculates an estimation for r_{ij}.

5. Edge-to-Edge Pricing Scheme

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

$$\hat{p}_{ij} = \frac{\hat{b}_{ij}}{c_{ij}} \qquad (3)$$

Here, $\hat{b}_{ij}$ represents user demand and c_{ij} is the available supply.

In Appendix C, we provided a detailed optimization analysis of this EEP pricing scheme in Distributed-DCC framework. We showed that the price calculation formula in Eq. (3) is optimal for the well-known total user utility maximization problem. We considered effect of different utility functions and elasticities of users on optimal prices.

6. Distributed-DCC: PFCC architecture

In order to adapt Distributed-DCC to PFCC architecture, LPS must operate on very low time-scales. In other words, LPS interval must be small enough to maintain control over congestion, since PFCC assumes no underlying congestion control mechanism. This raises two issues to be addressed:

- In order to maintain human involvement into the system, intermediate agents between customers and Ingress stations must be implemented.
- Since LPS must operate at very small time-scales, scalability issues regarding LPS must be solved.

As we previously said earlier in Section 3.1, we do not focus on the first problem since it cannot be addressed within this paper because of its large size and complexity. So, we assume that customers are willing to undertake high price variations, and leave development of necessary intermediate agents for future research. We address the second problem in Section 6.1.

6.1. 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 in 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.

7. Distributed-DCC: POCC architecture

In this section, we develop necessary components in order to adapt Distributed-DCC framework to POCC architecture. First, we will briefly describe an edge-to-edge congestion control mechanism Riviera [23]. Then, we will address problems defined in Section 3.2 for the case of overlaying Distributed-DCC over Riviera. This will fit Distributed-DCC to the POCC architecture.

Table 2
Differences between Distributed-DCC's PFCC and POCC versions

Distributed-DCC: PFCC	Distributed-DCC: POCC
LPS must operate at small time-scales	LPS may operate at large time-scales
LPS must be scaled because of its time-scale	It is not necessary to scale LPS
Framework can achieve a range of fairness in rate allocation	Fairness of rate allocation depends on the underlying congestion control mechanism
Bottleneck queues at network core are large	Bottleneck queues at network core are small
Does not need to manage queues at network edges	Need to manage queues at network edges

Also, to summarize Section 6 and this section, Table 2 shows differences between Distributed-DCC's PFCC and POCC versions.

7.1. Edge-to-Edge Congestion Control: Riviera

We now describe overall properties of an edge-to-edge congestion control scheme, Riviera [23], which we will also use in our experiments later in the paper.

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 (AIMD-Explicit Rate, 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. Every ingress node i maintains an additive increase parameter, α_i, 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_i) 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 Fig. 9a). If we want flow 1 to get a capacity of w times more than flow 2, then the following conditions must be hold:

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

7.2. Distributed-DCC over Riviera

We now provide solutions defined in Section 3.2, for the case of overlaying Distributed-DCC over Riviera:

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 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 communicate γ_{ij}s to the egresses, and they can multiply max_thresh_{ij} and min_thresh_{ij} with γ_{ij}.
2. *Edge queues*. In Distributed-DCC, ingress stations are informed by LPS about allocated capacity c_{ij} for each edge-to-edge flow. So, one intuitive way of making sure that the user will not contract for more than c_{ij} 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 updates the allocated capacity c_{ij} for flow i to j by the following formula $c'_{ij} = c_{ij} - Q_{ij}/T$, and uses c'_{ij} for price calculation. Note that Q_{ij} is the edge queue

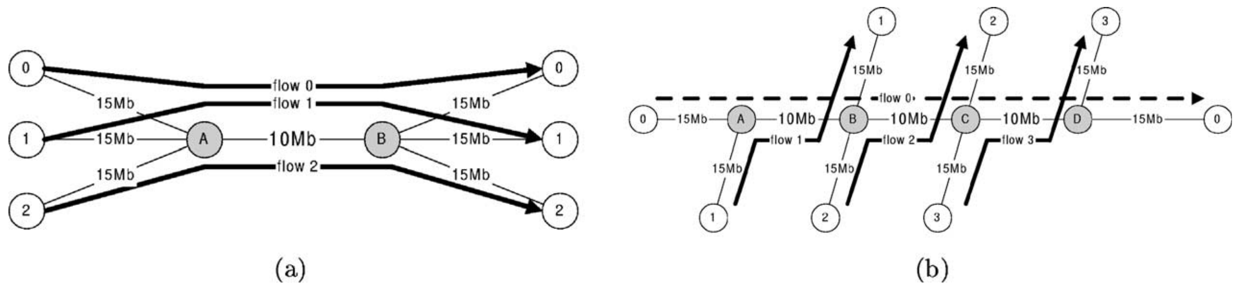


Fig. 9. (a) Single-bottleneck; (b) multi-bottleneck network for Distributed-DCC experiments.

length for flow i to j and T is the length of the contract.

An optional technique is as follows. Remember that egress nodes reduce their capacity estimation for a flow to a fraction of its current output rate, when a marked packet was received in the last observation interval. So, when edge queue exceeds a threshold, the ingress provider station can mark the packets, which will indirectly reduce the capacity estimation, and hence drain the edge queue.

8. Simulation experiments and results

We now present *ns* [24] simulation experiments for the two architectures, PFCC and POCC, on single-bottleneck and multi-bottleneck topology. Our goals are to illustrate fairness and stability properties of the two architectures with possible comparisons of two.

For PFCC and POCC, we simulate Distributed-DCC's PFCC and POCC versions which were describe in Sections 6 and 7, respectively. We will simulate EEP pricing scheme at Ingress stations. List of items we will present in the simulation experiments:

- Steady-state properties of PFCC and POCC architectures: queues, rate allocation
- PFCC's fairness properties: Provision of various fairness in rate allocation by changing the fairness coefficient α
- Performance of Distributed-DCC's capacity allocation algorithm ETICA in terms of adaptiveness.

8.1. Experimental configuration

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 10 Mb/s and all other links have 15 Mb/s. Propagation delay on each link is 5 ms, and users send UDP traffic with an average packet size of 1000 B. 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. Fig. 9a shows a single-bottleneck topology with $n=3$. Fig. 9b 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 truncated-Normal [25] 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 4 s, observation interval is 0.8 s, and LPS interval is 0.16 s. 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 4.4.3) 25 LPS intervals equivalent to one contracting interval.

The parameter δ is set to 1 packet (i.e. 1000 B), the initial value of $\hat{c}_{ij}$ for each flow f_{ij} is set to 0.1 Mb/s, β is set to 0.95, and Δr 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.

8.2. Experiments on single-bottleneck topology

We run simulation experiments for PFCC and POCC on the single-bottleneck topology, which is represented in Fig. 9a. In this experiment, there are three users with budgets of 30, 20, 10, respectively, for users 1, 2, 3. Total simulation time is 15,000 s, and at the beginning only the user 1 is active in the system. After 5000 s, the user 2 gets active. Again after 5000 s at simulation time 10,000, 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 7.2.

In terms of results, the volume given to each flow is very important. Figs. 10a and 11a show the volumes given to each flow in PFCC and POCC, respectively. We see the flows are sharing the bottleneck capacity in proportion to their budgets. 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 7.

Figs. 10b and 11b show the price being advertised to flows in PFCC and POCC, respectively. As the new users join in, the pricing scheme increases the price in order to balance supply and demand.

Figs. 10c and 11c show 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

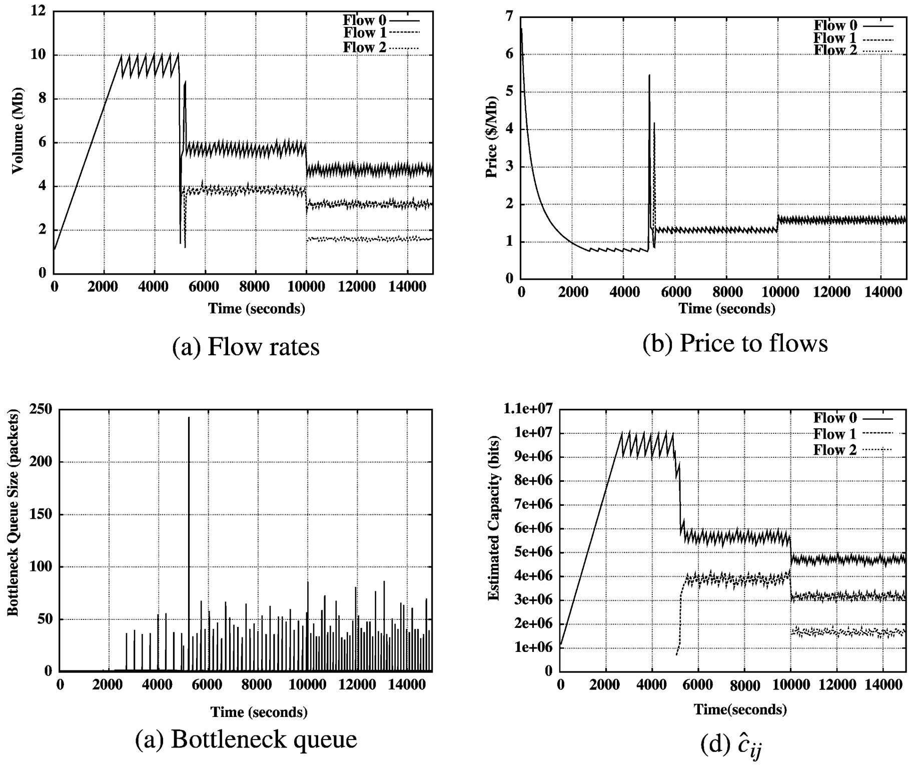


Fig. 10. Results of single-bottleneck experiment for PFCC.

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.

Fig. 12a–c shows the sizes of edge queues in POCC. We can observe that users get active at 5000 s of intervals. We observe stable behavior but with oscillations larger than the bottleneck queue shown in Fig. 11c. This is because of

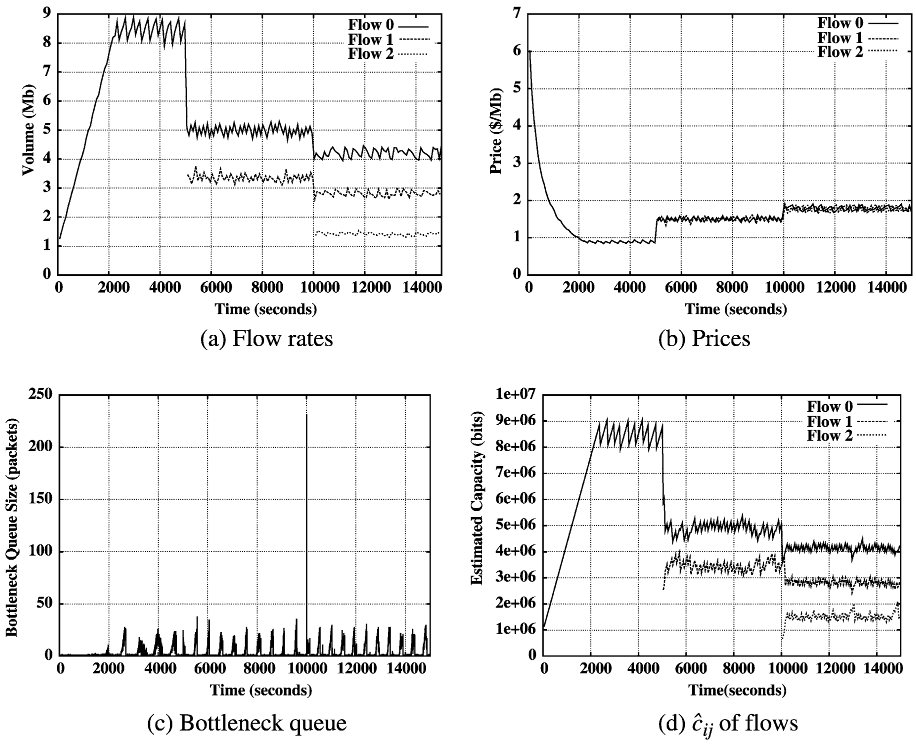
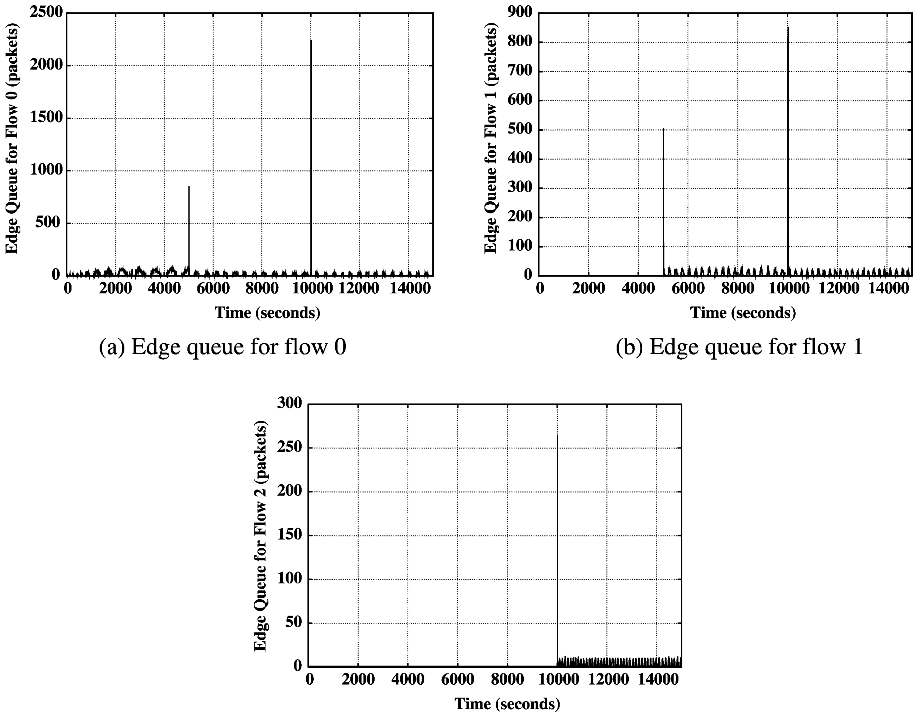


Fig. 11. Results of single-bottleneck experiment for POCC.



(a) Edge queue for flow 0

(b) Edge queue for flow 1

(c) Edge queue for flow 3

Fig. 12. Edge queues in the single-bottleneck experiment for POCC.

the tight edge-to-edge congestion control, which pushes backlog to the edges. This suits to the big-picture of the two architectures shown in Fig. 1.

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.

8.3. 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 4.4.4)
- *Property 2*: performance of Distributed-DCC's capacity allocation algorithm ETICA in terms of adaptiveness (see Section 4.4.3).

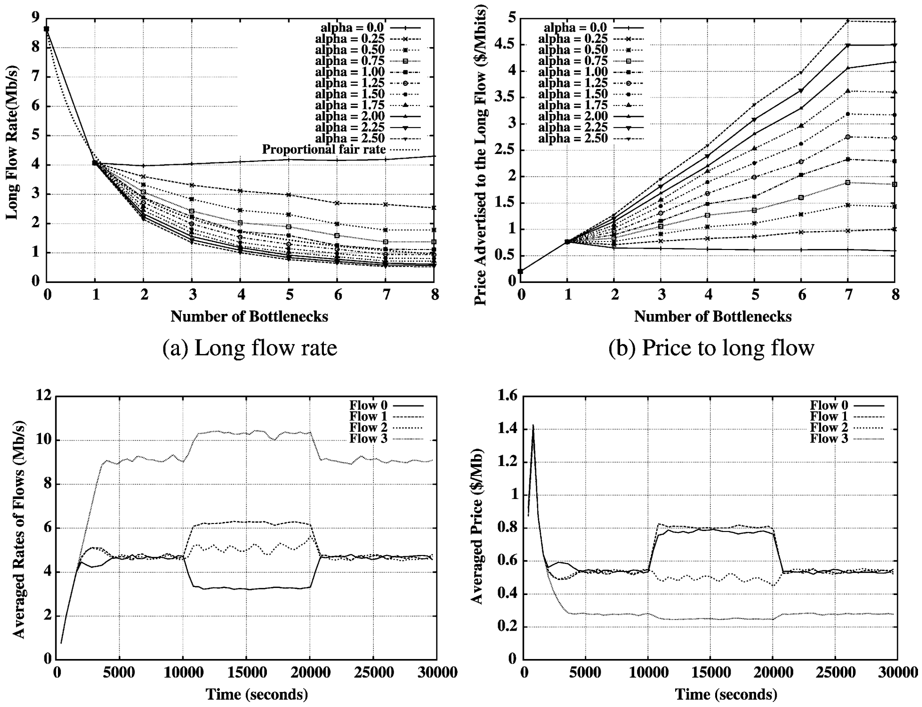
Since Riviera does not currently[4] provide a set of parameters for weighted allocation on multi-bottleneck topology, we will not run any experiment for POCC on multi-bottleneck topology.

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 Fig. 9b. In the multi-bottleneck topology there are 10 users and 9 bottleneck links. Total simulation time is 10,000 s. At the beginning, the user with the long flow is active. All the other users have traffic flows crossing the long flow. After each 1000 s, 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.

Fig. 13a shows the average volume given to the long flow versus the number of bottlenecks in the system for different values 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. Fig. 13b 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.

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



(a) Long flow rate

(b) Price to long flow

(c) Flow rates

(d) Prices to flows

Fig. 13. Results of PFCC experiments on multi-bottleneck topology.

Finally, to illustrate Property 2, we ran an experiment on the topology in Fig. 9b with small changes. We increased capacity of the bottleneck at node D from 10 to 15 Mb/s. There are four flows and three bottlenecks in the network as represented in Fig. 9b. Initially, all the flows have an equal budget of 10. Total simulation time is 30,000 s. Between times 10,000 and 20,000, 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 Section 8.2.

Fig. 13c shows the volumes given to each flow, and Fig. 13d shows the given volumes averaged over 200 contracting periods. Until time 10,000 s, 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 4.4.3).

Between times 10,000 and 20,000, 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 20,000, flows restore to their original volume allocations, illustrating the adaptiveness of the scheme.

9. Summary and discussions

In this paper, we presented a new framework, Distributed-DCC, for congestion pricing in a single diff-serv domain. Distributed-DCC can provide a contracting framework based on short-term contracts between user application and the service provider. Since contracts are short-term, it becomes possible to update prices frequently and hence to advertise dynamic prices. Particularly, on a totally edge-to-edge basis, we described ways of calculating congestion-based prices, which enables congestion pricing in the proposed Distributed-DCC framework.

Main contribution of the paper is to develop an *easy-to-implement* congestion pricing framework which provides flexibility in rate allocation. We investigated fairness issues within Distributed-DCC and illustrated ways of achieving a *range of fairness types* (i.e. from max-min to proportional) through congestion pricing under certain conditions. The fact that it is possible to achieve various fairness types within a single framework is very encouraging. We also developed a pricing scheme, EEP, within the Distributed-DCC framework, and presented several simulation experiments of it.

By extensive simulations, we demonstrated that Distributed-DCC's edge-to-edge capacity allocation algorithm, ETICA, has dominant effects of Distributed-DCC's performance especially when ratio of flows' budgets R is large.

Also, we introduced two pricing architectures based on the manner of attacking the problem of congestion control by pricing: PFCC and POCC. We adapted the Distributed-DCC framework to these architectures, and compared the architectures by simulation. We demonstrated that POCC is better in terms of managing congestion in network core, while PFCC achieves wider range of fairness types in rate allocation.

Future work should include investigation of issues related to extending Distributed-DCC on multiple diff-serv domains. One immediate question is that how will the end-to-end service be priced? One way of doing this is to make the Distributed-DCC domain end user is connected to responsible for provisioning of end-to-end service. Another way could be to design brokers that offer end-to-end services by buying edge-to-edge services from many Distributed-DCC domains at the background.

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 each individual user to contract up to the whole network capacity. This sometimes (especially when new users join in) causes 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 4.4.2), better budget estimation techniques (see Section 4.4.1).

Acknowledgements

Authors would like to thank Anuj Goel and Ritesh Pradhan for their valuable discussions and feedback.

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

Appendix A. Algorithm for Routing-sensitive Bottleneck-count Estimation (ARBE)

Given a diff-serv network, we would like to estimate number of bottlenecks each edge-to-edge flow is passing through. The algorithm ARBE presented in this appendix provides a solution to this problem.

Assuming that interior routers increment *bottleneck-count* header field of packets when congested, ARBE calculates the number of bottlenecks an edge-to-edge flow is passing through. ARBE operates at the *egress* edge router.

Assuming that each bottleneck has the same amount of congestion and also assume that they have the same capacity. Let $r_{ij}(t)$ be the number of bottlenecks the flow from ingress i to egress j, f_{ij}, is passing through at time t. ARBE operates on deterministic time intervals, and calculates $r_{ij}(t)$ as follows:

$$r_{ij}(t) = \begin{cases} \hat{r}_{ij}(t), & r_{ij}(t-1) \le \hat{r}_{ij}(t) \\ r_{ij}(t-1) - \Delta r, & \text{otherwise} \end{cases} \quad \text{(A1)}$$

where $\hat{r}_{ij}(t)$ is the highest number of bottlenecks that flow passed through in time interval t, Δr is a pre-defined value. $\hat{r}_{ij}(t)$ is updated at each packet arrival by simply equating it to the maximum of its actual value and the bottleneck-count header field of the newly arrived packet. Algorithm 1 shows the pseudo-code for the algorithm.

Algorithm 1. *Algorithm for Routing-Sensitive Bottleneck-Count Estimation*

ARBE($BC(t)$, Δr)
{Δr is decaying step-size.}
{$BC(t)$ is the maximum bottleneck-count received in the last interval t.}
{BC is the actual estimation for bottleneck-count.}
if $BC(t) > BC$ **then**
  $BC \leftarrow BC(t)$
else
  $BC \leftarrow BC - \Delta r$
end if

Realize that the bottleneck-count 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 bottleneck-count 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 Δr is) the value of r_{ij} will decrease to the actual number of bottlenecks the flow is passing through.

Appendix B. Max–min fairness, proportional fairness, and social welfare maximization

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 Fig. 9b. 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). 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:

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

$$Z = w_0 \log(x_0) + nw_1 \log(x_1) - nk(x_0 + x_1) + \lambda(x_0 + x_1 - C)$$

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

$$x_0 = \frac{w_0 C}{w_0 + nw_1} \qquad x_1 = \frac{nw_1 C}{w_0 + nw_1}$$

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. Refs. [8,26].

Appendix C. Optimization analysis of Edge-to-Edge Pricing

In Section 5, we described a pricing scheme EEP, which suits to the Distributed-DCC framework. The main idea of the EEP is to balance supply and demand by equating price to the ratio of users' budget (i.e. demand) B by available capacity C. Based on that, we used the pricing formula:

$$p = \frac{\hat{B}}{\hat{C}} \tag{A2}$$

where $\hat{B}$ is the users' estimated budget and $\hat{C}$ is the estimated available network capacity. The capacity estimation is performed based on congestion level in the network, and this makes the EEP scheme a congestion-sensitive pricing scheme (see Section 4.4.2).

In this appendix, we will provide theoretical proof that Eq. (A2) is optimal in the case of *logarithmic* user utilities. Further we will also show how to calculate optimal prices in the case of *non-logarithmic*[5] concave utilities.

We will also investigate users' elasticity to price and bandwidth. Specifically, we will first define different types of user elasticities, and then look at effect of these elasticities on optimal prices.

Also, note that optimization problem being solved is based on the assumption that each link in the network has an associated local price, just like in Low et al.'s [9] pricing framework. Notice that this violates the fundamental design principles of Distributed-DCC framework. This means our optimization study of EEP here is theoretically correct while Distributed-DCC framework trades off some optimality for implementation purposes.

C.1. Problem formulation

We now formulate the problem of *total user utility maximization* for a multi-user multi-bottleneck network.

Let $F = \{1, \ldots, F\}$ be the set of flows and $L = \{1, \ldots, L\}$ be the set of links in the network. Also, let $L(f)$ be the set of links the flow f passes through and $F(l)$ be the set of flows passing through the link l. Let c_l be the capacity of link l. Let λ be the vector of flow rates and λ_f be the rate of flow f. We can formulate the total user utility maximization problem as follows:

SYSTEM :

$$\max_{\lambda} \sum_f U_f(\lambda_f)$$

subject to

$$\sum_{f \in F(l)} \lambda_f c_l, \quad l = 1, \ldots, L \tag{A3}$$

This problem can be divided into two separate problems by employing monetary exchange between user flows and the network provider. Following Kelly's [27] methodology we split the system problem into two:

The first problem is solved at the user side. Given accumulation of link prices on the flow f's route, p^f, what is the optimal sending rate in order to *maximize surplus*.

[5] Note that non-logarithmic does not mean convex utility functions. Our proofs are valid only for concave utility functions.

$FLOW_f(p^f)$:

$$\max_{\lambda_f} \left\{ U_f(\lambda_f) - \sum_{l \in L(f)} p_l \lambda_f \right\} \tag{A4}$$

over

$$\lambda_f \geq 0 \tag{A5}$$

The second problem is solved at the provider's side. Given sending rate of user flows (which are dependent on the link prices), what is the optimal price to advertise in order to *maximize revenue*.

$NETWORK(\lambda(p^f))$:

$$\max_p \sum_f \sum_{l \in L(f)} p_l \lambda_f$$

subject to

$$\sum_{f \in F(l)} \lambda_f \leq c_l, \quad l = 1, \ldots, L$$

over

$$p \geq 0 \tag{A6}$$

Let the total price paid by flow f be $p^f = \sum_{l \in L(f)} p_l$. Then, solution to $FLOW_f(p^f)$ will be:

$$\lambda_f(p^f) = U_f' - 1(p^f) \tag{A7}$$

When it comes to the $NETWORK(\lambda(p^f))$ problem, the solution will be dependent on user flows utility functions since their sending rate is based on their utility functions as shown in the solution of $FLOW_f(p^f)$. So, in the next sections we will solve the $NETWORK(\lambda(p^f))$ problem for the cases of logarithmic and non-logarithmic utility functions.

C.2. Optimal prices: logarithmic utility functions

We model customer i's utility with the well-known function[6] [8,9,22,28]

$$u_i(x) = w_i \log(x) \tag{A8}$$

where x is the allocated bandwidth to the customer and w_i is customer i's budget (or bandwidth sensitivity).

Now, we set up a vectorized notation, then solve the revenue maximization problem $NETWORK(\lambda(p^f))$ described in the previous section. Assume the network includes n flows and m links. Let λ be row vector of the flow rates (λ_f for $f \in F$), P be column vector of the price at each link (p_l for $l \in L$). Define the $n \times n$ matrix P^* in which the diagonal element P^*_{jj} is the aggregate price being advertised to flow j (i.e. $p^j = \sum_{l \in L(j)} p_l$) and all the other elements are 0. Also, let A be the $n \times m$ routing matrix in which the element A_{ij} is 1 if ith flow is passing though jth link and the element A_{ij} is 0, if not, C be the column vector of link capacities (c_l for $l \in L$). Finally, define the $n \times n$ matrix $\hat{\lambda}$ in which the diagonal element $\hat{\lambda}_{jj}$ is the rate of flow j (i.e. $\hat{\lambda}_{jj} = \lambda_j$) and all the other elements are 0.

Given the above notation, relationship between the link price vector P and the flow aggregate price matrix P^* can be written as:

$$AP = P^* e \tag{A9}$$

$$\lambda = (\hat{\lambda} e)^{\mathrm{T}} = e^{\mathrm{T}} \hat{\lambda} \tag{A10}$$

where e is the column unit vector.

We use the utility function of Eq. (A8) in our analysis. By plugging Eq. (A8) in Eq. (A7) we obtain flow's demand function in vectorized notation:

$$\lambda(P^*) = WP^{*-1} \tag{A11}$$

where W is row vector of the weights w_i in flow's utility function (A8). Similarly, we can write derivative of Eq. (A11) as:

$$\lambda'(P^*) = -W(P^{*2})^{-1} \tag{A12}$$

Also, we can write the utility function (A8) and its derivative in vectorized notation as follows:

$$U(\lambda) = W \log(\hat{\lambda}) \tag{A13}$$

$$U'(\lambda) = W\hat{\lambda}^{-1} \tag{A14}$$

The revenue maximization of Eq. (A6) can be re-written as follows:

$$\max_P R = \lambda AP$$

subject to

$$\lambda A \leq C^{\mathrm{T}}. \tag{A15}$$

So, we write the Lagrangian as follows:

$$L = \lambda AP + (C^{\mathrm{T}} - \lambda A)\gamma \tag{A16}$$

where γ is column vector of the Lagrange multipliers for the link capacity constrain.

By plugging Eqs. (A11) and (A12) in appropriate places, the optimality conditions for Eq. (A16) can be written as:

$$L_\gamma : C^{\mathrm{T}} - WP^{*-1} \tag{A17}$$

$$L_{P^*} : -W(P^{*2})^{-1}P^*e + WP^{*-1}e - W(P^{*2})^{-1}A\gamma = 0 \tag{A18}$$

By solving Eq. (A18) for P^*, we obtain:

$$P^* = 0 \tag{A19}$$

Now, solve Eq. (A17) for P^* :

$$P^* = A(C^{\mathrm{T}})^{-1}W \tag{A20}$$

Apparently, the optimization problem has two solutions as shown in Eqs. (A19) and (A20). Since Eq. (A19) violates the condition $P > 0$, we accept the solution in Eq. (A20).

We finally derive P by using Eq. (A9):

$$P = (C^{\mathrm{T}})^{-1}We \tag{A21}$$

[6] Wang and Schulzrinne introduced a more complex version in Ref. [15].

Since $P^* = (P^*)^T$, we can derive another solution:

$$P = A^{-1} W^T C^{-1} A^T e \tag{A22}$$

Notice that the result in Eq. (A21) holds for a single-bottleneck (i.e. single-link) network. In non-vectorized notation, this results translates to:

$$p = \frac{\sum_{f \in F} w_f}{c}$$

The result in Eq. (A22) holds for a multi-bottleneck network. This result means that each link's optimal price is dependent on the routes of each flow passing through that link. More specifically, the optimal price for link l is accumulation of budgets of flows passing through link l (i.e. $W^T A^T$ in the formula) divided by total capacity of the links that are traversed by the flows traversing the link l (i.e. $A^{-1} C^{-1}$ in the formula). In non-vectorized notation, price of link l can be written as:

$$p_l = \frac{\sum_{f \in F(l)} w_f}{\sum_{f \in F(l)} \sum_{k \in L(f)} c_k}$$

C.3. Elasticity

The term *elastic* was first introduced to the networking research community by Shenker [29]. Shenker called applications that adjust their sending rates according to the available bandwidth as 'elastic applications', and the traffic generated by such applications as 'elastic traffic'. An example of such traffic is the well-known TCP traffic, which is adjusted according to the congestion indications representing decrease in the available bandwidth. Shenker, further, called applications that do not change their sending rates according to the available bandwidth as 'inelastic'. So, this interpretation of *elasticity* is the same as *adaptiveness*, i.e. an application is elastic if it adapts its rate according to the network conditions, it is inelastic if it does not.

The concept of elasticity originates from the theory of economics. In economics, demand elasticity according to price[7] is defined as *percent change in demand in response to a percent change in price* [30]. In other words, demand elasticity is the responsiveness of the demand to price changes. A formal definition of demand elasticity can be written as [30]:

$$\varepsilon = \frac{\Delta X(p)/X(p)}{\Delta p/p} \tag{A23}$$

[7] Demand elasticity can be defined according to several things other than price (e.g. time of service, delay of service). In the rest of the text, we will refer to demand elasticity to price when we say demand elasticity.

where p is price, Δp is the change in the price, $X(p)$ is user's demand function, and $\Delta X(p)$ is the change in user's demand. Eq. (A23) can be re-written as:

$$\varepsilon = \frac{p}{X(p)} \frac{dX(p)}{dp} \tag{A24}$$

Given ε, the characteristic L_ε of user demand is made according to the following functional definition [30]:

$$L_\varepsilon = \begin{cases} \text{elastic}, & |\varepsilon| > 1 \\ \text{unit elastic}, & |\varepsilon| = 1 \\ \text{inelastic}, & |\varepsilon| < 1 \end{cases}$$

So, Shenker's interpretation of elasticity for user utility is actually different from the real meaning of elasticity in economics. Note that Shenker defined elasticity of user utility (or application utility) according to bandwidth, let's call it ϵ. Let $u(x)$ be user's utility if he is given x amount of bandwidth. Then, following the argument in Eq. (A24), we can write ϵ as:

$$\epsilon = \frac{x}{u(x)} \frac{du(x)}{dx} \tag{A25}$$

According to Shenker's interpretation, the functional definition for L_ϵ (i.e. characteristic of user's utility according to bandwidth) will be as follows:

$$L_\epsilon = \begin{cases} \text{inelastic}, & \epsilon = 0 \\ \text{elastic}, & \epsilon \neq 0 \ \&\ \text{concave utility} \\ \text{not defined}, & \epsilon \neq 0 \ \&\ \text{convex utility} \end{cases}$$

Obviously, L_ϵ is a lot different than L_ε. Basically, L_ε interprets elasticity as *responsiveness* while L_ϵ does it as *adaptiveness*.

We can construct the relationship between ε and ϵ, given that the user solves the well-known maximization problem:

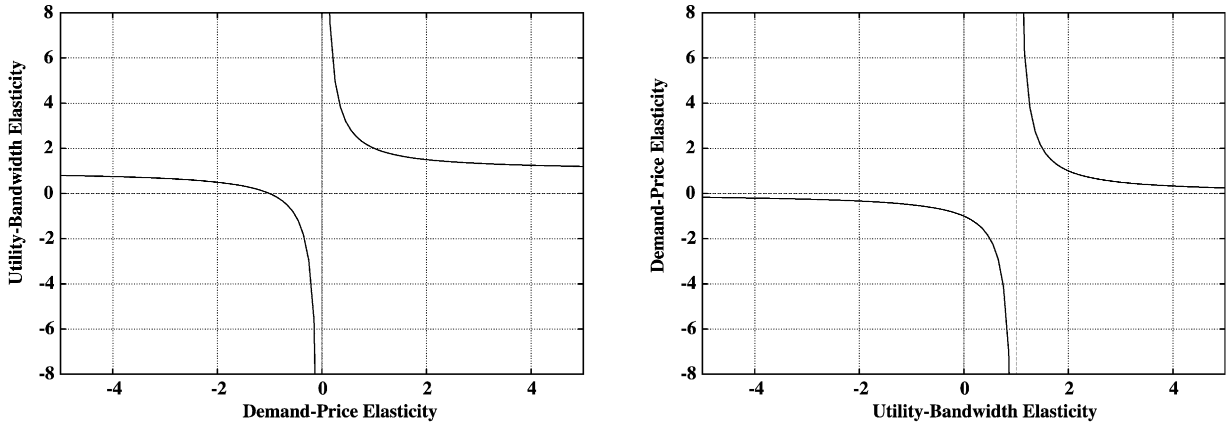
$$\max_x \{u(x) - xp\}$$

The solution to the above problem is $u'(x) = p$. So, given a price p, the user selects his demand such that his marginal utility equals to p. Based on that relationship between the utility function $u(x)$ and the demand function $X(p)$, we can construct the relationship between the demand-price elasticity ε and the utility-bandwidth ϵ elasticity. In the next sub-sections we will formulate the relationship between these elasticities.

C.3.1. Utility-bandwidth elasticity ϵ

Let $X(p) = Ap^\varepsilon$ where $\varepsilon \neq 0$ and $\varepsilon \neq -1$. Then,

$$p = u'(x) = A^{-1/\varepsilon} x^{1/\varepsilon}, \qquad u(x) = A^{-1/\varepsilon}\left(\frac{1}{\varepsilon} + 1\right) x^{1/\varepsilon + 1}$$



(a) Utility-bandwidth elasticity ϵ.

(b) Demand-price elasticity ε.

Fig. A1. Utility-bandwidth elasticity ϵ and demand-price elasticity ε with respect to each other.

So,

$$\epsilon = \frac{1}{\varepsilon} + 1, \quad \varepsilon \neq 0 \ \& \ \varepsilon \neq -1$$

Fig. A1a plots ϵ with respect to ε.

C.3.2. Demand-price elasticity ε

Let $u(x) = Bx^{\epsilon}$ where $\epsilon \neq 1$. Then,

$$u'(x) = p = A\epsilon x^{\epsilon - 1}, \qquad X(p) = \left(\frac{p}{A\epsilon}\right)^{1/(\epsilon-1)}$$

So,

$$\varepsilon = \frac{1}{\epsilon - 1}, \quad \epsilon \neq 1$$

Fig. A1b plots ε with respect to ϵ.

C.4. Optimal prices: non-logarithmic utility functions

In Section C.2, we derived optimal prices for the revenue maximization problem $NETWORK(\lambda(p^f))$. In that derivation users demand-price elasticity ε was -1 (see Eq. (A11)), which means users had *unit elastic* demands. Now, we re-perform the derivation by assuming that users have a utility-bandwidth elasticity of ϵ, where users' demand-price elasticity is $\varepsilon = 1/(\epsilon - 1)$ based on the study in the previous section. Also, note that $0 < \epsilon < 1$ must be satisfied in order to make sure concavity of the utility function.

First, let B be row vector of the weights that are different for each flow's utility function, and $\hat{B}$ be an $(n \times n)$ matrix in which the element $\hat{B}_{jj}$ is the weight of flow j and all the other elements are zero.

We use a generic utility function. The function and its derivative is as follows:

$$U(\lambda) = B\hat{\lambda}^{\epsilon} \tag{A26}$$

$$U'(\lambda) = B\epsilon\hat{\lambda}^{\epsilon-1} \tag{A27}$$

According to the relationship between ϵ and ε described in Section C.3.1, we can write the demand function and its derivative as follows:

$$\lambda(P^*) = \epsilon^{-\varepsilon} e^{\mathrm{T}} \hat{B}^{-\varepsilon} P^{*\varepsilon} \tag{A28}$$

Similarly, we can write derivative of Eq. (A28) as:

$$\lambda'(P^*) = \epsilon^{-\varepsilon} \varepsilon e^{\mathrm{T}} \hat{B}^{-\varepsilon} P^{*\varepsilon - 1} \tag{A29}$$

For the revenue maximization problem, we again solve the Lagrangian in Eq. (A16) but for the new demand function of Eq. (A28). By plugging Eqs. (A28) and (A29) in appropriate places, the optimality conditions for Eq. (A16) can be written as:

$$L_{\gamma} : C^{\mathrm{T}} - \epsilon^{-\varepsilon} e^{\mathrm{T}} \hat{B}^{-\varepsilon} P^{*\varepsilon} A = 0 \tag{A30}$$

$$L_{P^*} : \epsilon^{-\varepsilon} \varepsilon e^{\mathrm{T}} \hat{B}^{-\varepsilon} P^{*\varepsilon-1} (P^* e - A\gamma) + \epsilon^{-\varepsilon} e^{\mathrm{T}} \hat{B}^{-\varepsilon} P^{*\varepsilon} e = 0 \tag{A31}$$

By solving Eq. (A31) for P^*, we obtain:

$$P^* = \frac{1}{\epsilon} A\gamma e^{-1} \tag{A32}$$

Now, apply Eq. (A32) into Eq. (A30) and we get:

$$\frac{1}{\epsilon} A\gamma e^{-1} = \epsilon A^{-1/\varepsilon} (C^{\mathrm{T}})^{1/\varepsilon} (e^{\mathrm{T}})^{-1/\varepsilon} \hat{B} \tag{A33}$$

Substitute Eq. (A33) into Eq. (A32) and we obtain P^* :

$$P^* = \epsilon A^{-1/\varepsilon} (C^{\mathrm{T}})^{1/\varepsilon} (e^{\mathrm{T}})^{-1/\varepsilon} \hat{B} \tag{A34}$$

From Eq. (A34) we obtain P :

$$P = \epsilon A^{-1} A^{|1/\varepsilon|} ((C^{\mathrm{T}})^{|1/\varepsilon|})^{-1} (e^{\mathrm{T}})^{|1/\varepsilon|} \hat{B} e \tag{A35}$$

$$P = \epsilon A^{-1} A^{|1/\varepsilon|} ((C^{\mathrm{T}})^{|1/\varepsilon|})^{-1} (e^{\mathrm{T}})^{|1/\varepsilon|} (\hat{B}^{|\varepsilon|})^{|1/\varepsilon|} e \tag{A36}$$

The result in Eq. (A35) implies the same thing as in the case of logarithmic utility functions except that the link capacities must be taken more conservatively depending on the elasticity (ϵ or ε by choice) of flows. Observe that as flows demand-price elasticity ε gets higher, the capacity must be taken more conservatively based on the formula $(C^{\mathrm{T}})^{|1/\varepsilon|}$. Also observe that as flows utility-bandwidth elasticity ϵ gets higher, the capacity must be taken more conservatively based on the formula $(C^{\mathrm{T}})^{|1/\varepsilon|} = (C^{\mathrm{T}})^{|\epsilon-1|}$.

Based on Eq. (A36) we can write the optimal price formulas for single-bottleneck and multi-bottleneck cases, respectively, as follows in non-vectorized form:

$$p = \epsilon \left(\frac{\sum_{f \in F} w_f^{|\varepsilon|}}{c} \right)^{|1/\varepsilon|}, \qquad p_l = \epsilon \left(\frac{\sum_{f \in F(l)} w_f^{|\varepsilon|}}{\sum_{f \in F(l)} \sum_{k \in L(f)} c_k} \right)^{|1/\varepsilon|}$$

References

[1] J.K. MacKie-Mason, H.R. Varian, Pricing the Internet, Kahin, Brian and Keller, James, 1993.

[2] X. Wang, H. Schulzrinne, An integrated resource negotiation, pricing, and QoS adaptation framework for multimedia applications, IEEE Journal of Selected Areas in Communications 18, 2000.

[3] N. Semret, R.R.-F. Liao, A.T. Campbell, A.A. Lazar, Pricing, provisioning and peering: dynamic markets for differentiated Internet services and implications for network interconnections, IEEE Journal of Selected Areas in Communications 18, 2000.

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

[5] M. Yuksel, S. Kalyanaraman, B. Sikdar, Effect of pricing intervals on congestion-sensitivity of network service prices, Technical Report ECSE-NET-2002-1, Rensselaer Polytechnic Institute, ECSE Networks Lab, 2002.

[6] S. Blake, O. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, An architecture for Differentiated Services, IETF RFC 2475.

[7] R. Singh, M. Yuksel, S. Kalyanaraman, T. Ravichandran, A comparative evaluation of Internet pricing models: smart market and dynamic capacity contracting, Proceedings of the Workshop on Information Technologies and Systems (WITS) (2000).

[8] F.P. Kelly, A.K. Maulloo, D.K.H. Tan, Rate control in communication networks: shadow prices, proportional fairness and stability, Journal of Operations Research Society 49 (1998) 237–252.

[9] S.H. Low, D.E. Lapsley, Optimization flow control—I: basic algorithm and convergence, IEEE/ACM Transactions on Networking 7 (6) (1999) 861–875.

[10] A.M. Odlyzko, The economics of the Internet: utility, utilization, pricing, and quality of service, Technical Report, AT&T Labs, 1998.

[11] A.M. Odlyzko, Internet pricing and history of communications, Technical Report, AT&T Labs, 2000.

[12] I.C. Paschalidis, J.N. Tsitsiklis, Congestion-dependent pricing of network services, IEEE/ACM Transactions on Networking 8 (2) (2000) 171–184.

[13] A. Gupta, D.O. Stahl, A.B. Whinston, in: McKnight, Bailey (Eds.), Priority Pricing of Integrated Services Networks, MIT Press, Cambridge, 1997.

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

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

[16] A.M. Odlyzko, A modest proposal for preventing Internet congestion, Technical Report, AT&T Labs, 1997.

[17] D. Clark, in: McKnight, Bailey (Eds.), Internet Cost Allocation and Pricing, MIT Press, Cambridge, 1997.

[18] D. Clark, A model for cost allocation and pricing in the Internet, Technical Report, MIT, 1995.

[19] R. Cocchi, S. Shenker, D. Estrin, L. Zhang, Pricing in computer networks: Motivation, formulation and example, IEEE/ACM Transactions on Networking 1.

[20] M. Yuksel, S. Kalyanaraman, Simulating the Smart Market pricing scheme on Differentiated Services architecture, Proceedings of the GLOBECOM (2001).

[21] G.S. Arora, M. Yuksel, S. Kalyanaraman, T. Ravichandran, A. Gupta, Price discovery at network edges, Proceedings of the International Symposium on Performance Evaluation of Telecommunication Systems (SPECTS) (2002).

[22] S. Kunniyur, R. Srikant, End-to-end congestion contro: utility functions, random losses and ecn marks, Proceedings of the Conference on Computer Communications (INFOCOM) (2000).

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

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

[25] H.R. Varian, Estimating the demand for bandwidth, MIT/Tufts Internet Service Quality Economics Workshop (1999).

[26] D.M. Chiu, Some observations on fairness of bandwidth sharing, Technical Report TR-99-80, Sun Microsystems Labs, September 1999.

[27] F.P. Kelly, Charging and rate control for elastic traffic, European Transactions on Telecommunications 8 (1997) 33–37.

[28] J. Mo, J. Walrand, Fair end-to-end window-based congestion control, IEEE/ACM Transactions on Networking 8 (5) (2000) 556–567.

[29] S. Shenker, Fundamental design issues for the future Internet, IEEE Journal on Selected Areas of Communications 13 (1995) 1176–1188.

[30] H.R. Varian, Intermediate Microeconomics: A Modern Approach, W.W. Norton and Company, 1999.

Murat Yuksel is a Post-Doctoral Researcher and Lecturer at ECSE Department of Rensselaer Polytechnic Institute, Troy, NY. He holds a BS degree in Computer Engineering from Ege University, Izmir, Turkey in 1996. He received MS and PhD degrees in Computer Science from Rensselaer Polytechnic Institute in 1999 and 2002, respectively. His research is on various networking issues such as pricing, routing in wireless networks, large-scale network simulation, networking with free-space optics and performance analysis.

Shivkumar Kalyanaraman is an Associate Professor at the Department of Electrical, Computer and Systems Engineering at Rensselaer Polytechnic Institute in Troy, NY. He received a BTech degree from the Indian Institute of Technology, Madras, India in July 1993, followed by MS and PhD degrees in Computer and Information Sciences at the Ohio State University in 1994 and 1997, respectively. His research is in topics such as congestion control architectures, quality of service (QoS), free-space optical networking, network management, multicast, pricing, multimedia networking, and performance analysis. He was selected in 1999 by MIT's Technology Review Magazine as one of the 'TR-100", 100 young innovators for the new millennium. He is an associate member of IEEE and ACM.