# Dynamic Pricing and Traffic Engineering for Timely Inter-Datacenter Transfers

Virajith Jalaparti, Ivan Bliznets, Srikanth Kandula, Brendan Lucier, Ishai Menache
Microsoft

## ABSTRACT

As more business moves to the cloud, inter-datacenter bandwidth becomes an ever more valuable and congested resource. This bandwidth is typically sold using a fixed price per GB, and transfers are scheduled using traffic engineering mechanisms. However, this separation between the economic and engineering aspects of the problem makes it difficult to steer customer demand to lightly loaded paths and times, which is important for managing costs (typically proportional to peak usage) and providing service guarantees.

To address these issues, we design and evaluate Pretium – a framework that combines dynamic pricing with traffic engineering for inter-datacenter bandwidth. In Pretium, users specify their required rates or transfer sizes with deadlines, and a price module generates a *price quote* for different guarantees (*promises*) on these requests. The price quote is generated using internal prices (which can vary over time and links) which are maintained and periodically updated by Pretium based on history. A supplementary schedule adjustment module gears the agreed-upon network transfers towards an efficient operating point by optimizing time-varying operation costs. Using traces from a large production WAN, we show that Pretium achieves up to 80% of the social welfare of an offline oracular scheme, significantly outperforming usage-based pricing alternatives.

## CCS Concepts

**•Networks → Network economics; Data center networks;**

## Keywords

Inter-datacenter networks; dynamic pricing; percentile pricing; deadline scheduling.

*SIGCOMM '16, August 22 - 26, 2016, Florianopolis , Brazil*

DOI: http://dx.doi.org/10.1145/2934872.2934893

## 1 Introduction

Bandwidth on the wide area network (WAN) is a valuable and important resource. Private WANs, such as the ones connecting the datacenters at Google and Microsoft, carry both (i) user-facing traffic, which typically requires low latency, and (ii) large transfers of business data, which typically have deadlines. Centralized traffic engineering (TE) techniques have been proposed to improve network utilization [18, 20] without affecting low latency traffic and with explicit support for deadlines [22, 34]. Such techniques crucially depend on detailed traffic information: the priority class of requests, precise latency requirements and/or transfer deadlines, etc. In principle, these features would be provided by the users, but simply asking for this information can have unintended side-effects. Users are incentivized to inflate priorities and tighten deadlines, hoping for better service. The cumulative effect can drastically reduce the overall performance of the system.

Interviews with internal WAN customers at Microsoft (see Table 1 for a summary) reinforce that customers have strict requirements on many transfers and would pay more for guaranteed deadlines. Unfortunately, such mechanisms often do not exist today. The issue is prevalent also in public WANs, where customers pay for connectivity to cloud providers (e.g., Azure, Google ElasticCloud). The currently dominant method for pricing cloud bandwidth is fixed pricing (Table 2), which does not permit service guarantees such as deadline support for premium customers.

Another (seemingly unrelated) problem in the development of public WANs is the disconnect between provider costs and the most common pricing models. A large WAN provider can incur annual costs measured in hundreds of millions of dollars. The structure of these costs can be complex and non-linear: bandwidth costs often depend on the $95^{th}$ percentile usage of a link, and static capacity costs depend on peak utilization (since the network must be provisioned for such). However, the fixed pricing method currently in use provides no lever to lower peak usage specifically, and no lever to incentivize users to shift demand away from times of peak load.

We address these problems simultaneously with a framework that combines traffic engineering with dynamic pricing. Our framework has the following goals:

1. Each user should be presented a menu of prices up

front, from which they can select a service level.

2. Prices should align the preferences of individual users with the goals and costs of the platform.
3. Prices should be dynamic, adapting to changes in demand and usage patterns.
4. The pricing model should complement, not preclude, centralized traffic engineering.

We achieve these goals with a system called Pretium. Pretium serves both *byte requests* (e.g., move 10TB from DC$_1$ to DC$_2$ before 2AM) and *rate requests* (e.g., a firm leases 100 VMs in Azure US East and wants 250Mbps bandwidth guaranteed in/out of that datacenter for the duration of the lease). In Pretium, a customer specifies basic information within the request, and is immediately presented with a *price quote* that provides a menu of service levels (e.g., bandwidth guarantees) and corresponding prices. The customer can then choose a satisfactory point on the menu or modify the request if nothing is appropriate. The selected transfer is then managed by a central scheduler, tasked with upholding the service level guarantees. A key technical challenge in the design of Pretium is the *online* nature of the problem: price quotes must be generated as requests arrive, and be crafted to steer the system toward good aggregate performance while managing the incentives of individual customers.

To maintain high system efficiency while respecting user incentives, Pretium combines multiple ideas. First, it updates prices dynamically using a feedback loop. Pretium stores internal prices per link per time, which are used to generate each customer's price quote. Prices are updated periodically using the observed requests, closing the loop. Updates are based on dual pricing, and recent advances in combinatorial market design and statistical learning [6, 8, 19]. Second, Pretium plans well into the future so it can effectively balance price and service guarantees. It delays binding promised user traffic to the underlying paths until necessary, which allows for rerouting to accommodate later arrivals, link failures and congestion. Third, Pretium resolves the challenge of optimizing complex usage-based costs. Widely-used measures like 95$^{th}$ percentile usage (across time) of a link are hard to express as a tractable optimization problem. We show that the sum of top 10% usages is a good proxy for 95$^{th}$ percentile usage, and offer a way to solve the resulting scheduling problem as a succinct LP.

The net effect is that Pretium is able to efficiently allocate its network capacity, as measured by *social welfare*, i.e., the total value generated (over all requests served) minus operating costs.[1] Our experiments with traffic traces from a large production WAN show that Pretium achieves 60-80% of the social welfare relative to an aggressive benchmark: an omniscient offline optimization scheme that has full knowledge about future demands including their values[2]. In contrast, baselines that have oracular information but are restricted to using fixed prices achieve much lower social welfare and profit for the provider. We also find that Pretium outperforms alternative extensions to simple fixed prices, such as two-level pricing (peak vs. off-peak) and demand-driven spot market pricing. We also confirm (both theoretically and through simulations) that Pretium incentivizes reasonable and desirable behavior from customers.

| Questions | Responses |
|---|---|
| Do any transfers require deadlines? | 100% said yes |
| What fraction of transfers have strict deadlines? | 60% (on average) |
| Incur penalty on missed strict deadlines? | 88% said yes |
| Would you pay extra for guaranteed deadlines? | 64% said yes |
| Willing to delay some transfers if price reduced? | 81% said yes |
| Willing to use a network that guarantees deadlines, even if price is known only at the start of a transfer? | 81% said yes |

**Table 1:** Summary of results from a survey of several WAN customers at Microsoft.

Our conclusion is that it is necessary to intertwine pricing decisions with TE to best utilize WAN bandwidth. We present one implementation of such an integrated system, Pretium, and verify that it achieves much higher social welfare than the current practice of setting a static pricing policy and performing TE independently. Our contributions are:

- A novel system for network transfers on the WAN that combines online dynamic pricing and TE.
- By dynamically adapting prices and planning well into the future, Pretium offers a priori price quotes for *guaranteed* service (§4.1). A TE scheme at the back-end ensures that the service guarantees are met despite online arrivals and other network changes.
- A new way to model percentile-based usage costs as a compact set of linear inequalities (§4.2).
- An evaluation on the topology and traffic observed on a production WAN of a large enterprise (§6).

## 2 Background and Motivation

In this section, we provide necessary background for the motivation and challenges behind Pretium. For this purpose, we will refer to the characteristics of traffic on the inter-datacenter WAN of a large enterprise. We also conducted a user-survey of 15 operators and consumers of the WAN (Table 1).

**Lack of service guarantees.** Public cloud providers charge their user's network traffic as shown in Table 2. In general, traffic leaving the WAN (to the Internet) is charged more than traffic that stays on the WAN. Bulk discounts are offered and costs vary with geography. Note that current pricing schemes are static in time and across users. Cloud providers do not provide guarantees on network bandwidth and/or transferring data within deadlines (i.e., no SLAs). The provider-user engagements for private WANs are not publicly known, in general. In our internal survey, we found no evidence of formal service-level enforcement, although more than half of the customers we interviewed were interested in some form of guarantees (see Table 1).

[1] This is the natural generalization of many commonly-used metrics, such as utilization maximization and congestion minimization, to a setting with utility-weighted requests and operation costs.

[2] The offline scheme solves an LP with the full benefit of hindsight (complete information), and is nearly optimal. The deviation from optimality is due to linearizing the operation costs, which is required for tractability (§4).

| Cloud | Traffic billed | Price kind | | | Service kind | | Price (USD/GB) | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Dyn? | Geo. diff? | Bulk Discount? | Tiers? | SLAs | US/EU | Asia | South America |
| Amazon | between sites | No | No | No | No | No | 0-0.02 | 0-0.09 | 0-0.16 |
| | to Internet | No | Yes | Yes | No | No | 0.05-0.09 | 0.08-0.14 | 0.19-0.25 |
| Azure | to Internet | No | Yes | Yes | No | No | 0.05-0.09 | 0.12-0.14 | 0.16-0.18 |
| Google | to G prod. | No | Yes | No | No | No | 0-0.01 | | |
| | to Internet | No | Yes | Yes | No | No | 0.08-0.12 | 0.15-0.21 | 0.08-0.12 |
| Rackspace | all out | No | No | Yes | No | No | 0.06-0.12 | | |

**Table 2:** Pricing for WAN bandwidth by cloud providers (current as of 1/25/2016).

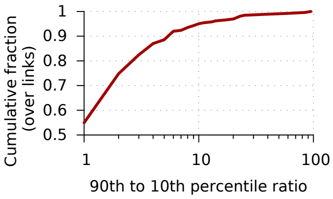


**Figure 1:** Ratio of $90^{th}$ percentile to $10^{th}$ percentile link utilization shown as a cumulative distribution function.

**Provider cost management.** Under the current state-of-the-art, providers can only imperfectly pass their costs to customers. Although TE algorithms exist to meet customer deadlines, there is no incentive for the customers to report true demands or deadlines. Hence, even if some users have flexible deadlines (Table 1), the lack of incentives for customers to move load to less busy periods forces the provider to over-provision network capacity (e.g., lease more capacity from the upstream ISPs to keep up with the peak demands).

One might argue that compute and storage are more expensive than network on the provider's balance sheet. However, getting the WAN pricing model correct has further implications: (i) the network can become a key differentiator across providers; and (ii) service guarantees on cross-site transfers are becoming increasingly important due to the demand for cloud-backed customer data and services (e.g., iCloud), and real time monitoring and management of services that have a global footprint (e.g., Facebook, Gmail).

**The need for adaptive pricing.** We find that a fixed price per link (or even a different price per hour-of-day) is unlikely to extract most of the gains of fully dynamic prices. Our analysis of data from a large inter-datacenter WAN shows sizable variation in the utilization of WAN links (Figure 1) – the $90^{th}$ percentile is more than 5 times the $10^{th}$ percentile utilization for more than 10% of the links, while it is less than 2 times for nearly 70% of the links. Though there is strong periodicity in the traffic, there are significant short-term variations in the volume, due to flash-crowds or link failures. The data also reveals that the extent of multiplexing in inter-datacenter networks is typically smaller than in Internet traffic, i.e., fewer transfers contribute substantial portions of the overall traffic. Consequently, careful resource-management of the WAN could make a difference; e.g., shifting a single large transfer by a few hours could accommodate many smaller competing transfers. To handle such demand variation, prices should react dynamically.

**Pricing versus auctions.** Pretium proceeds by quoting prices to customers upon arrival. An alternative design approach is to elicit bids, execute an auction, and then inform the customers of the outcome. Because of our online setting, standard auctions (such as a VCG auction) are not directly applicable; but there is a rich and relevant literature on incentive-aware online scheduling [4, 17] that can possibly be adapted to WAN allocation. Regardless of the feasibility of auctions, we advocate a pricing approach for two reasons. First, an auction forces customers to wait, potentially until their deadlines, to discover their price (or whether their traffic was routed at all). However, our user survey indicates that a priori guarantees on service and price are important (Table 1). Second, an auction requires users to fully specify and commit to the details of their request in advance, including all possible options of routes, durations, utility, etc. In contrast, Pretium responds with a menu of {price, service guarantee} options, allowing users to iteratively adjust their request if needed (e.g., to explore a different route or time window). Insofar as it is possible to achieve good performance and truthful behavior with a pricing approach, we view it as preferable to an auction.

**The need to combine traffic engineering with pricing.** Pricing can guard existing TE techniques that improve WAN utilization (e.g., [18, 20]) against strategic users. By quoting lower prices for more flexible requests (e.g., a request with a shorter deadline would be priced higher than a comparable request with a longer deadline), our pricing method encourages customers to report their true traffic requirements. Further, TE is an important tool for setting the right prices; without careful planning over paths and future times, some link-time pairs may appear more or less busy than they should be. A good TE procedure makes it easy to detect parts of the network that are overloaded and adjusts prices. Hence, Pretium carefully intertwines TE and pricing.

# 3 The WAN Ecosystem

In this section, we elaborate on the assumptions and notations of the economic model used in this paper. We then provide an example that highlights the benefits of Pretium.

## 3.1 Formal model

**The WAN.** We consider a bandwidth provider controlling a network $G$ of interconnected datacenters. Each edge $e = (u, v)$ in $G$ represents a WAN link between datacenters $u$ and $v$ or an egress link between datacenter $u$ and an ISP $v$. Link $e$ has associated capacity $c_e$, representing the total available bandwidth per unit-time on that link.

**The Customers.** There is a pool of users (customers) who make bandwidth requests. These requests arrive online. A

| Notation | Meaning |
|---|---|
| $(S_i, T_i)$ | Source and Target datacenters for request $i$ |
| $R_i$ | Set of all routes from $S_i$ to $T_i$ |
| $[t_i^1, t_i^2]$ | Timesteps when request $i$ is active (begin / deadline) |
| $v_i$ | Value per byte of request $i$ |
| $d_i$ | Total size (number of bytes) of request $i$ |
| $c_e$ | Total capacity of network link $e$ |
| $X_{irt}$ | # bytes of req. $i$ transmitted on route $r$ at time $t$ |
| $p_i(x)$ | Price to route $x$ bytes of data from request $i$ |
| $\pi_i(x)$ | Marginal price to route byte $x$ of request $i$ |
| $\bar{x}_i$ | Max. # of bytes from req. $i$ that can be guaranteed by its deadline |
| $x_i$ | Request $i$'s chosen amount of data |
| $g_i$ | The guaranteed transfer for request $i$: $\min\{x_i, \bar{x}_i\}$ |
| $B_{i\tau}$ | # bytes of request $i$ transferred by time $\tau$ |
| $C(X)$ | Platform cost of schedule $X = (X_{i,r,t})$ |
| $y_e$ | 95th percentile of bandwidth usage on link $e$ |
| $C_e$ | Cost per unit of 95th percentile usage on link $e$ |
| $W$ | Number of timesteps per time window |

**Table 3:** Summary of notation.

*byte request*, indexed by $i$, has a quantity of data $d_i$ to be routed. The request indicates the source $S_i$ and target $T_i$; data must be transmitted along a set of admissible paths (or *routes*) $R_i$ from $S_i$ to $T_i$. The request specifies a *time interval* in which the data can be routed. We discretize time into *timesteps*, where each timestep corresponds to, for example, a five-minute interval. Accordingly, the time interval of each request $i$ is translated to $[t_i^1, t_i^2]$, where $t_i^1$ is the timestep corresponding to the start time of the request, and $t_i^2$ is the timestep corresponding to its deadline. A sizeable portion of inter-datacenter transfers have deadlines, and can be modeled using this abstraction [18, 22]. For example, periodic index refreshes are expected to be fully available before a certain time of day. Some deadlines can be soft. Each request can potentially be routed along multiple paths ($|R_i| \geq 1$). While this can result in packet-level reordering, we do not model its effects, assuming existing techniques can be applied to resolve any reordering (e.g., [29]).

Some network transfers may not have a concrete deadline. A common scenario is transfers with a constant *rate* requirement for a certain period of time. Hence, Pretium supports *rate requests* in addition to byte requests. For brevity, we omit a detailed model of rate requests, but note that they can be handled as a special case of our solution (§4.4). Other portions of the WAN traffic may not be governed by any TE scheme [18, 20, 22]. For example, there could be some latency-sensitive requests or short transfers (e.g., in the order of few seconds) that have to be routed immediately without waiting for the TE solution. We call such traffic *high-pri* and set aside a portion of capacity into the future to accommodate such requests; see §4.4 for more details.

Each request has a value $v_i$, which is the most the customer is willing to pay per byte transferred. For simplicity, we assume linear values: the value of a partial transfer is proportional to the quantity of data transferred. Nevertheless, the design principles behind Pretium do not depend on a specific customer-utility model; in §4.4, we discuss how our solution can be extended for non-linear utilities.

Finally, we note that we deliberately do not make a clear distinction in this paper between *private* and *public* cloud WANs. Pretium can be applied in either setting. In a private cloud, the customers are individuals or groups within a large enterprise, each of whom could be allocated a budget for infrastructure usage (in either real or virtual currency), and are therefore incentivized to minimize WAN transfer costs. In a public cloud, Pretium allows the cloud provider to dynamically set its prices for WAN transfers.

**Costs.** The provider pays a cost to run the network. Operator interviews reveal that some of the WAN links are "private" or "owned", meaning that their cost only changes during capacity planning, which happens a few times each year. The remaining links are charged based on usage. Discussions with operators reveal that pricing based on 95th percentile usage is a common approach. These links are typically purchased from upstream providers and connect the WAN to the Internet. The 95th percentile utilization is typically calculated over a fixed time period, such as a day, week or month.

**Schedules.** The provider is in charge of routing/scheduling decisions, hence we use the terms "provider" and "scheduler" interchangeably. At each timestep, the scheduler must choose which data to transfer, and along which paths, subject to capacity constraints. Scheduling is done online: each request arrives at a certain time $a_i$ ($\leq t_i^1$), and the provider is unaware of requests before they arrive.

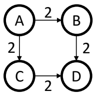
We write $X_{irt}$ for the number of bytes from request $i$ transmitted along route $r \in R_i$ at time $t$. The quantities $X = (X_{irt})$ fully describe a schedule of transfers. We write $C(X)$ for the total cost generated by schedule $X$. We defer the specific description of $C(X)$ to §4.2, and note that $C(\cdot)$ can be nonlinear, e.g., modeling 95th percentile charges.

**Prices and customer Behavior.** The network provider can charge payments to customers. In general, the price of a transfer can depend on the number of bytes transferred, the source and target nodes, the timing of the request, network load, etc. In the implementations we consider, the system exposes the payment rule to each customer in advance of any data transfers. Customers can then choose whether or not to route data at the offered prices and, if so, how much to transfer. Supposing that $x_i$ units of data from request $i$ are transmitted within the time interval $[t_i^1, t_i^2]$, and that the network provider charges a total price of $p_i(x_i)$ for this transmission, the *utility* of customer $i$ is taken to be $u_i(x_i) = v_i \cdot x_i - p_i(x_i)$. Each customer is a self-interested agent aiming to maximize utility.

**Objectives.** We focus primarily on the objective of system efficiency: generating a schedule that maximizes the total value across all customers, minus provider costs. Precisely, the system efficiency of schedule $X$ is

$$\sum_i \sum_{r \in R_i} \sum_{t \in [t_i^1, t_i^2]} v_i \cdot X_{irt} - C(X). \qquad (1)$$

This metric captures the overall efficiency of the network utilization. It is important that costs are accounted for directly in this objective: cost minimization is an important aspect of transfer planning, and it is inefficient to route traffic whose value does not at least outweigh its cost burden. We note that this objective is known in the economic literature as *social*



v: value per byte, D: demand

| Request | v | D | [start, end] |
|---|---|---|---|
| $R_1$: A → B | 8 | 2 | [0,1] |
| $R_2$: A → B | 4 | 2 | [0,2] |
| $R_3$: A → D | 4 | 2 | [0,1] |
| $R_4$: C → D | 1 | 4 | [0,2] |

| | $R_1$ | $R_2$ | $R_3$ | $R_4$ | Welfare |
|---|---|---|---|---|---|
| No Price | 1 | 2 | 1 | 3 | 23 |
| Fixed price: 4 | 1 | 2 | 2 | 0 | 24 |
| Fixed price per link:<br>(A → B: 8, B → D: 0, A → C: 2, C → D: 2) | 2 | 0 | 2 | 0 | 24 |
| Fixed price per timestep: ([0,1]: 4, [1,2]: 1) | 1 | 2 | 2 | 2 | 26 |
| Pretium | 2 | 2 | 2 | 2 | 34 |

**Figure 2:** Example illustrating the impact of pricing. Top: the network (all links have capacity 2 units) and request specifications. Bottom: the number of units of each request scheduled under different pricing methods (all prices per unit transferred), and total welfare of the schedule.

*welfare* [33], and the optimization in (1) is known as welfare maximization. Welfare is a natural objective for private networks, but is also highly relevant in public cloud settings: social welfare is a generalization of common performance metrics, such as utilization, to a setting with utility-weighted requests and operation costs.[3]

As a secondary objective, we also measure the profit of the network provider when evaluating Pretium. This is the sum of payments made by the customers minus costs incurred. The profit of a schedule $X$ is $\sum_i p_i(x_i) - C(X)$.

**Limitations.** Our model does not feature bulk discounts, which may help, e.g., in attracting new users to the cloud, and utilizing capacity that is already paid for but would go unused otherwise (e.g., [7]). It is reasonable to assume that bulk discounts would be less relevant in a highly-utilized network with transfer-SLAs, as in Pretium. Furthermore, we do not explicitly model competition between providers. That said, we do assume that the provider is concerned with maximizing the efficiency of its WAN resources, which may be motivated by competitive pressure from other providers [33].

## 3.2 Example

We use a simple example of a network with four nodes and four requests (Figure 2) to illustrate the importance of prices in WANs, and showcase the benefits of Pretium.

**Alternatives.** A system without payments cannot distinguish high-value requests from low-value ones, and the optimization problem becomes one of throughput maximization. As a result, only a fraction of the valuable requests ($R_1$ and $R_3$) is scheduled. Charging a fixed price $p$ per unit sent ensures admission control (customers with value less than $p$ will opt out) but is too coarse to target specific regions and time periods. In our example, a price of 4 per unit achieves the highest welfare but this causes $R_1$ and $R_2$ to share the link $(A, B)$, even though $R_2$ can be deferred to a later time.

Using different fixed prices on each link in the network, and charging requests the sum of prices on links traversed, allows for the use of higher prices in congested parts of the network, but still fails to respond to temporal shifts in demand. In this case, the optimal link prices are 8 on link $(A, B)$, 2 each on links $(A, C)$ and $(C, D)$, but these prices prevent requests $R_2$ and $R_4$ from transmitting in the second timestep when the system is underutilized. Alternatively, one could use a single uniform price but make it vary over time. For instance, one could set higher prices when the system is under heavy load. In our example, prices of 4 and 1 for the two timesteps achieve the best welfare but this again causes $(A, B)$ to be shared by the two requests $R_1$ and $R_2$.

**Pretium combines both spatial and temporal price differentiation.** In Pretium, each link is given a price at every timestep, and these prices can vary over time in response to observed usage patterns. In this example, Pretium could set a price of 8 on link $(A, B)$ for the first timestep but lower it to 4 in the second timestep. This allows $R_2$ to be deferred to a later time and allows $R_1$ (of higher value) to finish within its allocated deadline. Similarly, the price on link $(C, D)$ would be set to 4 in the first timestep and lowered to 1 (to allow $R_4$) in the second timestep. Overall, in this example, Pretium can achieve the maximum possible welfare of 34. The next section describes how Pretium determines the prices.

# 4 Design of Pretium

We now turn to a detailed description of Pretium. Pretium consists of three modules, as shown in Figure 3. A *network state* datastructure is central to the various modules of Pretium. This maintains the prices for the links and a plan for how to route the accepted requests. Both aspects are maintained for multiple timesteps into the future.

1. The *request admission interface* (RA) (§4.1) is the user interface. In response to a transfer request, it uses the network state to generate a *price quote* for routing different guaranteed amounts of request data. Customers can choose how much data they wish to transfer. A preliminary routing schedule (or plan) is chosen.
2. The *schedule adjustment module* (SAM) (§4.2) runs at each timestep to decide the actual routing and scheduling for that timestep. This module can update the plans for future timesteps. Its goal is to maximize welfare (total value minus cost) while respecting a priori guarantees on service (e.g., deadlines).
3. The *price computer* (PC) (§4.3) updates the prices for each link and for each future timestep. While RA updates prices after accepting individual requests, the PC module anticipates future load levels (based on historical usage and current demand levels) and performs a more holistic optimization to set prices.

Each of the above modules works at a different timescale. The request admitter (RA) responds with a price quote *online* to every new request. The RA executes a multi-timestep TE to identify the least expensive way to schedule different portions of the user transfer. Price quotes are computed by it in a manner that ensures truthful behavior from users (§4.1).

RA lets the user pick a price and a service guarantee. Then, once per timestep, the schedule adjustor (SAM) figures out the

[3] Another natural objective for public networks is profit maximization. However, we note that it is important to account for market competition when optimizing for profit. We take the position that the cloud market is highly competitive, and hence a profit-seeking provider will anyway be driven to optimize for welfare [33].

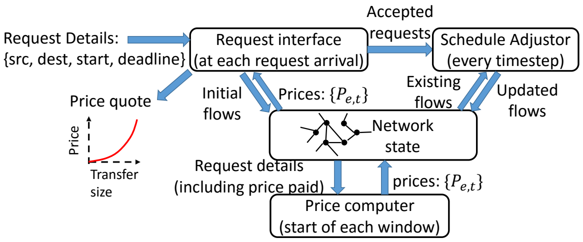


**Figure 3:** Overview of Pretium.

actual routes to install at switches and the amount of rate to allocate to each user on each route in order to achieve the promised service guarantees. SAM is related to recent work on multi-timestep traffic engineering [22, 34] but extends it in two ways: (1) it considers potentially non-linear link costs and (2) optimizes for social welfare. Further, by using prices, SAM is mostly protected from strategic users (§4.2). Similar to prior work [18, 20], we execute SAM once every few minutes. This leaves enough time to put routes in SDN switches and change rates at the sending servers.

Finally, once every *time window*, the price computer (PC) aggregates and analyzes all requests over a recent time period, and updates the baseline link prices. Our logic to set prices is straightforward (we use dual prices); however, our method to learn the anticipated load levels is novel (§4.3). We recommend prices be recomputed each hour.

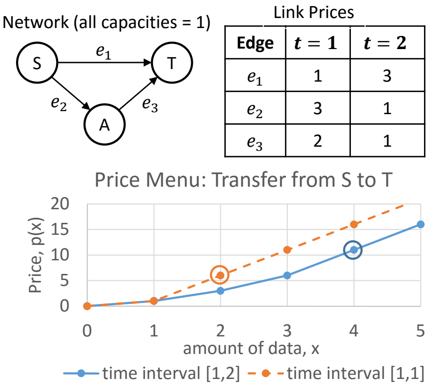
When building an intertwined TE + pricing scheme, we believe that decoupling the timescales is important. The three modules share the network state but otherwise function independently. A monolithic implementation may be less nimble in responding to the user than the RA. Further, delaying the actual scheduling and routes to SAM lets Pretium react to unexpected faults and congestion. We next describe the details of each of the modules.

## 4.1 Request Admission Interface

When a customer's request $i$ arrives, Pretium must determine how that request would be served. The customer will be quoted a price menu $p_i(\cdot)$, where $p_i(x)$ is the price to route $x$ bytes of data. This menu depends on the current state of the network and the request's parameters (source, destination, arrival time, and deadline). A capacity upper bound $\bar{x}_i$ is also reported to the customer: $\bar{x}_i$ is the maximum amount of data that Pretium guarantees can be routed by the deadline. The customer then chooses how much data, $x_i$, to transfer (if any). This decision is viewed as a contract: Pretium guarantees that $\min\{x_i, \bar{x}_i\}$ bytes will be routed within the specified time interval. If the customer asks to transfer more than $\bar{x}_i$, then data beyond $\bar{x}_i$ is routed on best-effort basis.

**Calculating the price quote.** The price computer (§4.3) maintains, at each point in time, a price-per-byte $P_{e,t}$ for each link in the network $e$ and each future time step $t$. For request $i$, $p_i(x)$ is defined to be the minimum total price (i.e., sum of edge prices) at which $x$ bytes can be routed within the allowed time interval.

There is a natural interpretation of $p_i(\cdot)$: first route traffic along the minimum-price path/time until the path is saturated – each byte is priced accordingly – then along the path with the next lowest price, and so on. As the price per byte only increases, the price schedule $p_i(\cdot)$ is a non-decreasing, convex, and piece-wise linear function (Figure 4). The fact that Pretium uses minimum-price paths in this calculation is important as it drives its incentive properties (§5). It also serves as a link between pricing and TE as lower-price paths will tend to be those that have less congestion (§4.3).



| Edge | *t* = 1 | *t* = 2 |
|---|---|---|
| $e_1$ | 1 | 3 |
| $e_2$ | 3 | 1 |
| $e_3$ | 2 | 1 |

**Figure 4:** Sample price menus for two different requests, differing only on their deadlines. A shorter deadline leads to higher prices. The maximum possible guarantees ($\bar{x}$) are circled.

**User response.** Given the price quote, each user decides how much demand to send. Write $\pi_i(x) = p_i(x) - p_i(x-1)$ for the marginal price to route byte $x$. The optimal choice for each customer is to route as much of their demand as possible while the marginal price is at most their value per byte (i.e., $\pi_i(x) \leq v_i$). We establish this formally in §5. In this paper, we assume that each request is associated with a fixed deadline. However, one may also envision users with flexible deadlines; e.g., "transfer my entire demand as soon as possible, as long as the price is not too high." Such preferences can be manifested by users resubmitting requests with different deadlines (each deadline would correspond to a different price quote). The analysis and evaluation of Pretium with flexible deadlines, and request rescissions is an interesting direction for future work.

**Preliminary schedule.** The customer's chosen transfer is immediately assigned a preliminary schedule, determined by the price computation. Bandwidth is reserved (over multiple timesteps, if needed) for the request on the minimum-price routes that were used to compute $p_i(x_i)$, and link utilization is updated accordingly. In this way, the admission interface also performs TE by steering traffic toward low-price paths.

**Capacity Bound $\bar{x}$.** During periods of high utilization, it may not be possible to route all requests fully. The price quote therefore includes a "maximum countably transfer," $\bar{x}_i$, the largest transfer that Pretium will guarantee by the specified deadline. The price menu extends beyond $\bar{x}_i$ (Figure 4) and if the user chooses a large value $x_i > \bar{x}_i$, only $\bar{x}_i$ data is guaranteed to be routed; any additional data (beyond $\bar{x}_i$) is routed on a best-effort basis, at a price of $\pi(\bar{x}_i)$. Allowing customers to specify demands beyond $\bar{x}_i$ is important for improving utilization, as additional data may be transferable after schedule adjustment if the amount of *high-pri* traffic is less than anticipated.

**Short-term price adjustments.** To increase robustness to sudden demand spikes, we increase link prices in response to heavy, localized congestion. This short-term adjustment complements the price computer (§4.3) which operates on a slower timescale and updates all link prices at once. We found that increasing the price of a link by a multiplicative factor if its utilization crosses a threshold is a simple yet efficient rule for controlling short-term demand spikes (e.g., double the price of the last 20% of the link capacity). This short-term adjustment is performed after each request is admitted by the RA module. We note that this price adjustment is functionally equivalent to splitting each network link into parallel links with different prices. Hence, the adjustment does not qualitatively change the complexity of the price menu and user response.

## 4.2 Schedule Adjustment

The admission interface determines how much data to transfer and performs initial traffic engineering, but the exact routing schedule is flexible. The schedule adjustment module exploits this flexibility by reoptimizing at each timestep.

**Problem formulation.** The schedule adjustor considers all requests whose full demand has not been satisfied and whose deadline has not expired. The objective is to maximize total welfare (values minus costs) while satisfying flow guarantees. This is a non-convex optimization problem, due to the non-convex costs. Write $B_{i\tau}$ for the total number of bytes transferred before time $\tau$ for request $i$, and $g_i = \min\{\bar{x}_i, x_i\}$ for the number of bytes guaranteed to be transferred (§4.1). As user values $v_i$ are unknown, the marginal price from the admission controller, $\pi_i \equiv \pi_i(x_i)$, is used as a proxy.[4] Formally, the objective at timestep $\tau$ is

$$\begin{aligned} \text{maximize} & \sum_i \sum_{\tau \le t \le t_i^2} \sum_{r \in R_i} X_{irt} \cdot \pi_i - C(X) \qquad (2) \\ \text{subject to} & \sum_{\tau \le t \le t_i^2} \sum_{r \in R_i} X_{irt} \le x_i - B_{i\tau} \quad \forall i \\ & \sum_{\tau \le t \le t_i^2} \sum_{r \in R_i} X_{irt} \ge g_i - B_{i\tau} \quad \forall i \\ & \sum_{i:\tau \le t \le t_i^2} \sum_{e \in r, r \in R_i} X_{irt} \le c_{e,t} \quad \forall t, e, \end{aligned}$$

The available capacity for each link $c_{e,t}$ can vary over time. As described in §3, the available capacity depends on the expected volume of high-priority traffic.

We emphasize that although (2) uses the marginal prices $\pi_i$ as a proxy for the expected values, the above program does not encode profit maximization. Intuitively, profit-maximizing prices are typically much higher, in order to extract more revenue from high-valued customers at the expense of lower utilization and overall user satisfaction.

**Non-convex costs.** The term $C(X)$ of (2) is potentially non-convex when link costs are usage-based. For example, 95$^{th}$ percentile charges can be modeled as $\sum_e C_e \cdot y_e$ where $C_e$ is an edge-specific constant and $y_e$ is the relevant 95$^{th}$ percentile usage on that link. This makes the optimization problem hard:

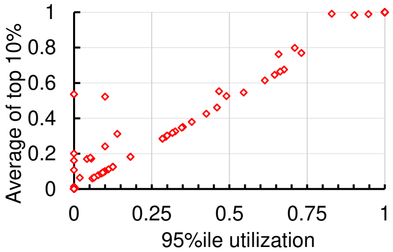


**Figure 5:** Scatter plot of 95$^{th}$ percentile and average of top 10% utilization values. Each point corresponds to a link.

THEOREM 4.1. *Maximizing (2), when $C(X)$ is non-convex, is an NP-hard optimization problem.*

The proof follows by a reduction from the subset-sum problem [28]; we omit the details for brevity.

**Solution.** We deal with the above challenge by using an alternate metric that approximates the true costs. Define $z_e$ to be the utilization on edge $e$, averaged over the 10% of time steps in the window with highest utilization. For example, if the window contains 30 timesteps and link $e$ was utilized most on steps 7, 13, and 26, then $z_e = \frac{1}{3}(X_{e,7} + X_{e,13} + X_{e,26})$, where $X_{e,t}$ is the total traffic routed along edge $e$ on step $t$. Intuitively, $z_e$ will be positively biased over the 95$^{th}$ percentile usage on edge $e$, $y_e$. The bias will be more significant for heavy-tailed traffic distributions. Using traffic data from our production network, we experimentally found that $z_e$ is linearly correlated with $y_e$, over different time periods. This relation is shown in Figure 5 as a scatter plot – each point represents a single link in the network. To validate this assumption further, we used a variety of synthetic distributions (normal, exponential and pareto) to model network traffic. For each distribution, we generated link traffic over time. Given these traffic samples, we calculated $z_e$ and $y_e$ for each link $e$, and found them to be linearly correlated, with small difference between the absolute values.

Following the above analysis, we can solve an approximation of (2) by substituting the 95$^{th}$ percentile cost by $\sum_e C_e \cdot z_e$. The resulting optimization problem can be encoded as a linear program, although the straightforward encoding requires exponentially many constraints (as there are exponentially many ways to choose 10% of the timesteps in a window). We address this issue by using sorting-network inequalities [25], which reduces the number of constraints to polynomial without loss in accuracy.

THEOREM 4.2. *There exists a set of $O(kT)$ linear constraints which expresses an upper bound on sum of top $k$ values from the set $X_{e,1}, \ldots X_{e,T}$.*

See the appendix for the construction and proof. We note that our solution improves upon the techniques proposed in [25] by requiring 40% fewer “sorting" constraints per link (details in appendix). Furthermore, we provide a rigorous proof of correctness, which was missing in [25].

[4] The use of marginal prices for estimating user values is motivated by Theorem 5.2, which implies that users will set these quantities equal if possible.

## 4.3 Price Computation

The price computer maintains temporal link prices $\{P_{e,t}\}$. A price is stored for every link, and for every timestep in the current *time window* (e.g., a day). We write $W$ for the number of timesteps in a time window. Typically, most request deadlines fall within the time window. To provide full flexibility for requests with deadlines beyond the current time window, we simply carry over the same prices to the following time windows. This approach works well when $W$ is chosen in accordance with the periodic pattern of demand.

Prices are periodically updated at the start of every time window using recent traffic data. The price computer considers a previous *reference window*. It takes as input all requests that arrived during a certain previous period of time (say $T$, of length at least $W$) that contains the reference window. It solves an offline version of the scheduling problem to calculate the optimal prices, in hindsight, for time period $T$. These prices, restricted to the reference window, are then used as the updated link prices.

We allow $T$ to be larger than the reference window because prices can be distorted at the beginning and end of period $T$ (since earlier requests are not included, and requests are not scheduled beyond $T$). Allowing $T$ to extend beyond the reference window reduces the impact of this distortion.

**Reference window selection.** There are multiple options for the reference window. One natural choice is the preceding window. If the demand follows a clear diurnal pattern and windows are shorter than 24 hours, one might instead choose the corresponding window from the day before. These simple choices performed well in our simulations (§6).

**Value estimation.** As with schedule adjustment, the price computer uses the marginal price-per-byte ($\pi_i$) chosen from the price menu as a proxy for the value of a request.

**Computing prices.** Given the input described above, the price computer encodes the offline welfare-optimization problem as an LP. It then solves the dual of that LP, which can be interpreted as assigning prices to each (link, timestep) pair. These prices, restricted to the reference window, will be used as the updated internal link prices.

This method of computing prices is self-correcting. To see why, suppose the price of a link was set too low. The RA would tend to admit more requests on that link as they will be offered lower prices. This causes increased congestion. When prices are next updated, the optimal offline schedule will address the congestion by diverting jobs away from that link, corresponding to an increase in its dual price. In this way, prices tend to rise for highly-demanded regions of the network at peak usage times. Similarly, if a link is priced too high, fewer requests will be admitted on that link. This reduces congestion and causes the price to fall upon recalculation. Price convergence is discussed further in §4.4.

## 4.4 Discussion

**Rate requests.** So far, we described Pretium under the assumption that users specify byte requests; i.e., transfer a certain number of bytes by a deadline. Pretium handles rate requests via the following extension: model a rate request as a sequence of byte requests, one per timestep. The request admission interface computes prices separately for each time step and quotes a total price by taking a sum over the entire interval. The schedule adjustment module ensures that the rate will be achieved in each timestep.

**Nonlinear utilities.** For simplicity, our model assumes that user utilities are linear in the number of bytes transferred. In practice, some users may have non-linear utilities, including all-or-nothing transfers. But we note that the core functionality of Pretium does not depend on this linearity assumption. In the request admission phase, users can choose a pricing option and transfer amount that maximizes their overall utility, regardless of the nature of their utility function. In particular, a user with low value for a partial transfer can elect not to choose a partial-transfer option from the price menu. If desired, users could also indicate that they are not interested in sending more traffic at the same price (e.g., if their utility function is concave), which would be respected by the schedule adjustment module. The main way that Pretium uses the assumption of linear utilities is in the price computer, which employs linearity during offline price optimization. However, even if customer utilities are not truly linear, we conjecture that prices computed under a linearity assumption will be approximately correct in practice if there are enough requests in the system (i.e., if no single request consumes a significant fraction of the network capacity). A precise analysis of Pretium when customers have non-linear utilities is a direction for future research.

**Network faults and unexpected increases in high-pri volume.** As described earlier, Pretium sets aside some capacity to account for ad hoc high priority traffic; the volume to be set aside is estimated based on historical usage [18]. When unexpected congestion occurs, perhaps because of more high-pri traffic or network faults, Pretium's schedule adjustment module tries to satisfy all guarantees by spreading the load over other paths and future times. In practice, we find that the likelihood of reneging on guarantees is small.

**Best-effort requests.** The contrary to the above is that there may be less-than-anticipated high-pri volume. As noted already, the schedule adjustor uses the residual capacity at each timestep to route more volume of accepted requests which in some cases lets requests finish before deadline. Though less likely, it is possible that more unused capacity remains. Hence, Pretium can offer a "scavenger" class wherein requests can choose their price and Pretium schedules them in a best-effort manner.

**Hybrid requests.** We note that the above changes allow users to purchase a guarantee $x$, either for bytes or a rate, and simultaneously make a scavenger class request. Such added flexibility allows users (and their applications) further flexibility to get good service at a low cost.

**Convergence and Stability of price choice.** In the most general case of arbitrary request sizes and arrivals, it is hard to prove strong properties about the online price selection method (§4.3) but one can do so under some simplifying assumptions. When requests are drawn from the same underlying distribution of customer demands and if there are sufficiently many requests per time window, the price selection will be approximately optimal for the upcoming window as well. See [6] and Theorem 52 of [19] for a formalization.

**Impact of dynamic prices on users.** In Pretium, the price of a request is unknown until it is submitted. We take the position that the service guarantee (i.e., achieving the deadline) offsets the drawback of price-uncertainty. Our survey of operators at Microsoft (Table 1) shows that there is demand for this tradeoff, and many are willing to use such a system.

**Fairness.** The main objective of Pretium is to achieve high social welfare, which balances total customer satisfaction and platform profit. However, Pretium does not explicitly enforce a fairness criterion. For example, a small number of customers with high value and large demand (e.g., large organizations in the public cloud setting) can drive up prices beyond the reach of users with less willingness to pay. To mitigate such scenarios, Pretium can be supplemented with constraints that limit the effect of elephant traffic. For example, the network provider could impose a limit on the amount of bandwidth that can be allocated to any one user.

**Interplay between the three modules.** We conclude by reaffirming the interplay between the three parts of Pretium. In the strictly offline case, when request parameters are known a-priori, the request admitter would work well alone by simply choosing *dual prices*. The modified price selection method (described above) helps with online request arrivals; it learns the prices based on estimated usage. Further, the schedule adjuster allows for high-pri traffic and offers robustness to unexpected faults or other congestion events.

# 5 Incentives and User Behavior

As discussed earlier, one motivation for Pretium is that in a TE-only setup, customers can receive better service by inflating their values or misrepresenting deadlines. We will show that by combining prices with TE appropriately, Pretium discourages such strategic behavior.

**Attack model.** Each user request consists of the following parameters: $\{S_i, T_i, R_i, t_i^1, t_i^2, d_i\}$. In order, these are the source, target, allowed routes, begin time, end time, and demand (in bytes). The customer's value (per byte), $v_i$, is private information not reported to Pretium. We allow the user to make changes to every aspect of his request, and even to break it into multiple requests between possibly different sources and targets. Under this attack model, we prove that:

THEOREM 5.1. *Each user request maximizes utility by reporting only a single request and by truthfully reporting above parameters (under some technical conditions).*

The proof, which appears in the appendix, follows from a standard monotonicity argument. Because the RA interface offers the lowest possible price for each request (by checking all the specified routes and time-periods) any deviation in parameters from their true values will either (a) lead to a no-better price for the request or (b) cause the request to receive a no-better service, or both.

There is one caveat, however, which necessitates the "technical conditions" in the theorem statement: if more capacity becomes unexpectedly available (e.g., because of an unanticipated dip in high-pri traffic), the schedule adjustor could choose to route some requests earlier than anticipated. This allows users to strategize. For example, a user could report a later deadline (possibly obtaining a lower price) and still *hope* to receive full service by the true deadline. However, we verified empirically that the potential gains of manipulation are low, by sampling users and simulating deviations. In typical executions, fewer than 26% of admitted requests could benefit (i.e., increase their utility) by altering their parameters even with omniscient knowledge of the system state, and the average improvement (conditional on being able to benefit) was less than 6%. Due to the risk of missing deadlines, we believe that users are unlikely to misreport for minor potential gains. We hence claim that:

CLAIM 1 (INFORMAL). *In practice, customers will maximize their expected utility by truthfully reporting the parameters of their requests.*

Further, we offer some constrained versions of Pretium (i.e., the technical conditions referred to in Theorem 5.1) that guarantee strategyproofness. One option is to disable the schedule adjustment module. However, we believe that such dynamic adaptation is useful both for robustness and to get good network utilization. A second option is to enforce that the transferred data can only be used *after* the stated deadline $t_i^2$ (e.g., if the user specifies a deadline of 5pm, then at least some portion of the data will be unavailable before 5pm even if the transfer completes ahead of schedule). This can be implemented at the network or the application layers by withholding some portions of the transfer until the stated deadline. Intuitively, this helps because the request can never finish before its stated deadline. In the appendix we prove Theorem 5.1 under the assumption that transfers are withheld until their deadlines. In practice, we feel that the unconstrained Pretium is sufficiently strategy-proof and do not recommend implementing these technical conditions.

Finally, we prove that users respond to price menus in a predictable manner, as described in §4.1.

THEOREM 5.2. *Given a quoted price schedule $p_i(\cdot)$, customer $i$ maximizes utility by choosing to route $\min\{d_i, \max\{x\colon \pi_i(x) \le v_i\}\}$ bytes of data.*

The proof (omitted due to lack of space) follows from the fact that it is in the user's interest to route as much data as possible, as long as the marginal price is at most $v_i$.

# 6 Evaluation

We evaluated Pretium using simulations by replaying traffic traces that were collected from a large production inter-DC WAN. Our main results are as follows.

(1) Pretium achieves more than 60% of the optimal welfare, which is 3.5X higher than a region-based fixed pricing scheme. Further, Pretium results in more than 2X higher profit for the provider.

(2) We demonstrate that the prices picked by Pretium adapt to load variations and that Pretium is robust to variations in network and request characteristics.

(3) By comparing across a wide set of alternatives, we show that combining TE with pricing offers clear benefits over performing either TE or pricing individually.

## 6.1 Methodology

**Datasets.** We collected a month-long traffic trace from a production inter-DC WAN. The network has 106 nodes and 226 edges; each node is a datacenter or a site. The collected traffic trace is sampled NetFlow data and it was difficult to group the various network flows into appropriate user requests. Hence, we convert the network data into a time-series of traffic matrices between datacenters. Based on operator survey about typical request parameters (size, average request duration, deadline, etc.), we generated requests that closely mimic the observed traffic matrix time-series, while using different distributions for individual values and deadlines (e.g., normal). We note that we do not explicitly model high-pri requests in our simulations. We assume that the bandwidth required for such requests is known a priori (e.g., from historical usage, as in [18, 22]), and is appropriately reserved on all links of the network. The link capacity available for Pretium is reduced by this reserved amount.

**Link costs.** Around 15% of the WAN edges are priced based on 95th percentile usage, calculated over a period of 24 hours. Other links have fixed installation costs which are not included in the social welfare formulation (§3.1).

**Load factor.** To examine the performance of Pretium under different levels of network load, we scale the traffic matrix with a *load factor*; higher or lower values indicate proportionally more or less load.

**Solver.** We built our modules as linear programs and execute them using the Gurobi solver [1].

**Metrics.** We evaluate a few different metrics. First, we use Equation 1 to compute the social welfare from carrying traffic i.e., total value minus costs. Second, we examine provider profits. Finally, we report related metrics such as network utilization, and fraction of requests that finish.

**Baselines.** We compare Pretium with various baselines.

(1) *Offline optimal (*`OPT`*).* We refer to `OPT` as the best tractable *offline* solution we could come up with. Specifically, `OPT` provides an upper bound on the welfare of any TE+pricing scheme that approximates 95th percentile costs using the scheme described in §4.2. It assumes knowledge of all future requests and their real values, and solves a linear program which aims to maximize the total welfare (Equation 1). We use the term `OPT` for simplicity. The true optimal welfare might be higher; however, we cannot compute it precisely due to non-convex 95th percentile costs (§4.2).

(2) *Offline scheduling without pricing (*`NoPrices`*).* This mimics state-of-the-art TE schemes that do not use prices. Recall that without prices users can strategize in several ways (change deadline, claim higher value or higher demand etc.). Rather than evaluating all strategic options, we consider the following simplification. As such, a scheduler cannot credibly learn the customer values, we offer it full information about requests except for the value. `NoPrices` solves a single offline LP to maximize the sum of total bytes transferred by the network (assume value=1 for every request) minus the cost incurred. Practical online versions of this scheme (such as Tempus [22]) would obviously perform worse and hence we do not consider them in our evaluation.

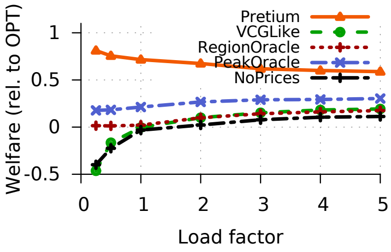


**Figure 6:** Welfare relative to `OPT` at different load factors.

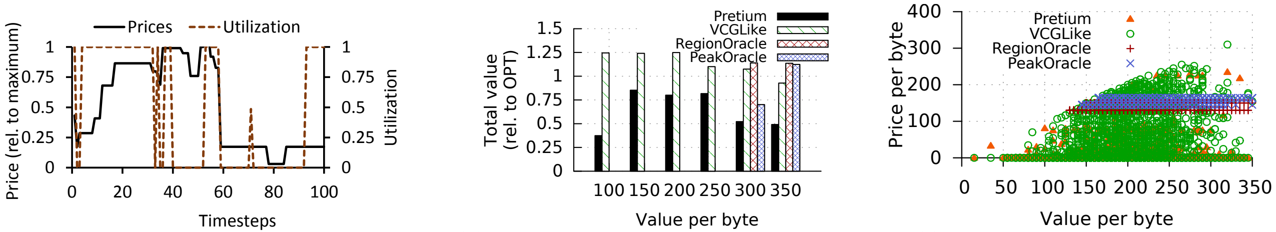
(3) *Region-based pricing oracle (*`RegionOracle`*).* This scheme closely resembles the prices used in practice (Table 2). The network is divided into a few regions (corresponding to U.S., Europe, Asia etc.). Requests within a region pay one price per byte and requests between regions pay a higher price. We try all possible values of these two price numbers, and choose the ones that offer the highest welfare in hindsight. These prices are used to admit requests, and admitted requests are scheduled so as to transfer the maximum amount of bytes before the respective per-request deadlines while accounting for the 95th percentile network costs.

(4) *Time-of-day based pricing oracle (*`PeakOracle`*).* In this scheme, we divide the day into two periods: (a) peak period, and (b) off-peak period. These periods are statically chosen based on the traffic traces – the peak period is chosen as the time interval when the network utilization is consistently over the daily average, and the remaining time is set as off-peak. Requests are charged a lower price during the off-peak period, and a higher price during the peak period. As in the case of `RegionOracle`, we set the prices optimally by searching the space of prices and selecting those that maximize welfare, in hindsight.

(5) `VCGLike` *spot pricing scheme (*`VCGLike`*).* This scheme models a spot market that responds to demand. Along with their requests, customers submit bids ($b_i$) which are assumed to be equal to the value of the requests ($v_i$). At each timestep, all byte requests are converted into rate requests by calculating the bandwidth $r_i$ required to route any remaining demand by the deadline. Then, the provider routes requests, up to their allotted bandwidth, to maximize the declared welfare ($\sum_i b_i r_i$). Finally, each customer is charged their VCG payment. This process is repeated every timestep. Even though VCG payments are used, this scheme is not truthful as byte requests are served over multiple timesteps. Further, it does not account for the costs incurred by the provider.

## 6.2 Benefits of Pretium

Figure 6 shows the welfare achieved by the above baselines and Pretium, relative to the offline optimal (`OPT`). The request values are drawn from a normal distribution with standard deviation smaller than the mean. We find that without prices (`NoPrices`), the welfare can end up being negative. This is because without prices any request can enter the system and there is no incentive to spread the load across lightly-loaded paths or time-periods. Hence, the network costs become high; much higher than the total value from bytes carried. While `RegionOracle` achieves 1-17% of the optimal welfare, `PeakOracle` achieves 18-30% of it. This



**(a)** Temporal variation of prices and utilization on a single network link.

**(b)** Total value achieved for different values per byte.

**(c)** Price charged vs. value of request.

**Figure 7:** Dynamic prices help increase welfare achieved by Pretium.

shows that pricing can increase the overall usefulness of the system. We note that `RegionOracle` and `PeakOracle` are oracular schemes rather than practical ones, as they use the true values of all requests to determine prices.

While the `VCGLike` baseline uses the users' bids (equal to their values) to schedule requests, it does not account for provider costs and results in negative welfare at lower load factors. Pretium outperforms all the other baselines and achieves more than 60% of the optimal welfare. The benefits of Pretium stem from using dynamic prices based on demand and network utilization. This is shown in Figure 7a where Pretium charges higher prices (black solid line) during periods of higher utilization (brown dotted line) on a particular link in the network, for a load factor of 2. Pretium prevents lower valued requests from occupying bandwidth that can be better used to serve requests with higher value.

Further, to illustrate the benefits of Pretium, Figure 7b shows a histogram of the total value achieved (y-axis) by different schemes relative to `OPT`, binned by the value per byte of the requests (x-axis). Correspondingly, Figure 7c shows the value per byte (x-axis) of each request and the price (y-axis) at which it is admitted into the network. `RegionOracle` and `PeakOracle` do not perform well as they set high prices to accept higher valued requests, and do not provide service to requests with low value (value from requests in lower buckets is 0). Pretium overcomes this problem with a richer price structure, resulting in requests with lower value being admitted at lower prices, and charging higher prices to requests with higher value. `VCGLike` retrieves more value from requests of lower value per byte but performs poorly in terms of welfare as it myopically allocates requests per timestep and ignores the costs incurred.

**Profits.** Figure 8 shows the profits of Pretium, `PeakOracle` and `VCGLike` for the provider, relative to `RegionOracle` – Pretium achieves more than 2X higher profits. This gap is higher at lower load factors, as `RegionOracle` sets a high price per byte (to offset the costs on edges) and only a few requests are admitted. As a result, a large portion of the network is under-utilized and the provider's revenue is low.

**Request completion.** Pretium achieves higher request completion compared to the baselines (Figure 9) because (a) unlike `VCGLike`, it plans into the future and (b) unlike `NoPrices`, `PeakOracle` and `RegionOracle`, it schedules requests with lax deadlines and smaller values during periods of low network load. We also note that Pretium provides completion guarantees to requests when admitted but none of the other schemes provide such guarantees.

| Module | Run time (sec) | |
|---|---|---|
| | Median | $95^{th}$%'ile |
| RA (for each request) | 0.59 | 0.62 |
| SAM (for every 5 minute timestep) | 0.99 | 1.08 |
| PC (for every 24 hour time window) | 3.17 | 3.3 |

**Table 4:** Runtimes of different modules in Pretium.

**Network utilization.** Pretium performs schedule adjustment every timestep to minimize the incurred costs (§4). This, in turn, reduces the spikes in link utilization. Figure 10 shows the CDF of the $90^{th}$ percentile link utilization for the different schemes. Pretium reduces the median link's $90^{th}$ percentile utilization by nearly 30% compared to `RegionOracle`. It further reduces the maximum utilization on about 16% links (not shown) by 25%, representing significant savings in WAN bandwidth.

**Benefits of different modules in Pretium.** The separation of price-adaption and traffic engineering (via schedule adjustment) in Pretium, along with the flexibility offered to requests (§4) all contribute towards its gains. Figure 11 shows the welfare (relative to `OPT`) of (a) Pretium without price menus (Pretium-NoMenu), in which all requests either get their entire demand allocated or none, and (b) Pretium without schedule adjustment (Pretium-NoSAM), where the schedule adjustment module in Pretium is skipped.

We see that the flexibility Pretium offers with a menu of prices allows it to achieve nearly 1.3X-2X higher welfare (Pretium vs. Pretium-NoMenu in figure). Further, Pretium-NoSAM performs more than 3X times worse than Pretium – a large portion of this difference is because of reoptimizing the schedule at every timestep, accounting for changes in network traffic and taking network costs into account significantly improves performance.

**Computational overhead.** Table 4 shows the execution time for each module in Pretium for the above setup. The request admission module (`RA`), which is on the critical path of every request, and the schedule augmentation module (`SAM`), which runs every timestep, are fast and take around a second. The price computer takes around 3 seconds but it runs once every 24 hours, and incurs a small overhead to the provider.

## 6.3 Sensitivity analysis

In this section, we explore the sensitivity of Pretium to variations in link costs, and request value distributions.

**Link costs.** We varied the mean link cost of the network links by over 2X (at a load factor of 1). The welfare (relative

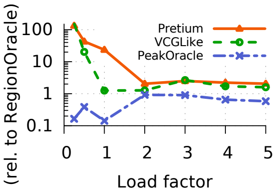


**Figure 8:** Profits of different schemes.

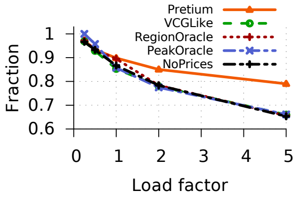


**Figure 9:** Of the requests admitted, fraction of requests completed.

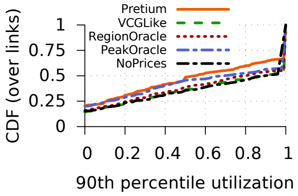


**Figure 10:** Network utilization.

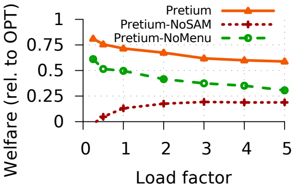


**Figure 11:** Benefits of different modules in Pretium.

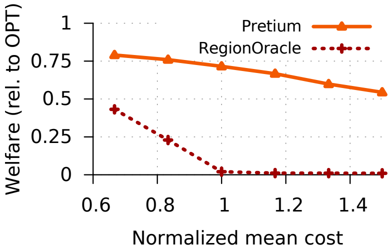
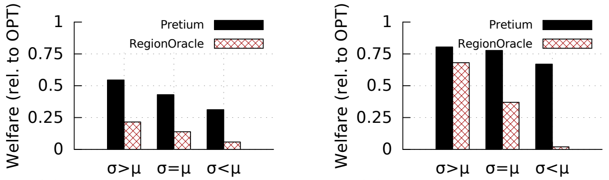


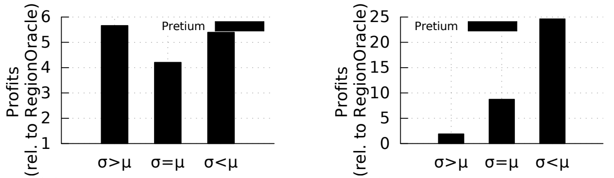
**Figure 12:** Varying mean link cost, at load factor 1.



**(a)** Pareto distribution.

**(b)** Normal distribution.

**Figure 13:** Welfare achieved for value distributions with different mean ($\mu$) to standard deviation ($\sigma$) ratios, at load factor 1.



**(a)** Pareto distribution.

**(b)** Normal distribution.

**Figure 14:** Profits of Pretium for value distributions with different mean ($\mu$) to standard deviation ($\sigma$) ratios, at load factor 1.

to OPT) obtained by Pretium and RegionOracle is shown in Figure 12. While the welfare of both schemes reduces with increasing cost, RegionOracle has a significantly higher reduction because it sets higher prices to compensate for the increasing cost. As a result, fewer requests are admitted. Pretium is more robust as it increases the prices only on the higher cost links and retains lower prices on low cost links.

**Request value distribution.** The benefits of Pretium depend on the distribution of request values. Figure 13 shows welfare of Pretium and RegionOracle (relative to OPT) where request values are drawn from (a) pareto and (b) normal distributions with different mean ($\mu$) to standard deviation ($\sigma$) ratios. Figure 14 shows the profits achieved by Pretium relative to RegionOracle for the same distributions. While both the metrics vary with the distributions, we see that Pretium consistently outperforms RegionOracle.

# 7 Related Work

Pricing for communication networks has been studied for over three decades; see [9] for an extensive survey. Initial applications were motivated by the ATM standard, followed by proposals to incorporate pricing for Internet congestion control (e.g., [23, 26, 30]) and inter-ISP transit networks [27, 31]. More recently, there has been work on increasing profits of ISPs through flexible pricing structures, leading to better utilization and user satisfaction. TUBE [16] deals with pricing mobile data. TUBE uses time-dependent dynamic pricing (TDP) instead of traditional per-usage pricing, to offload delay-tolerant users in peak traffic periods. Similar ideas can be found in [35]. Our work also incorporates time-varying prices, but our setting and interface are different in that requests specify deadlines and demands, and the provider uses traffic engineering to accommodate multiple requests. [32] proposes destination-based tiered pricing for transit ISPs – based on both the traffic demand, as well the cost of carrying it; they use three or four tiers. We also offer "tiers" for demand; however, the actual price also depends on current congestion levels and the delay tolerance of users. To the best of our knowledge, we are the first to propose and analyze a pricing framework for WAN traffic.

The Pretium pricing model and update method is motivated by the theory literature on dynamic pricing and machine learning. The existence of optimal "market" prices is a classical result in economics [3], and given sufficient data it is possible to learn those prices [6]. Hsu et al. [19] point out that such prices can be used to guide online allocation. An alternative line of work [5, 8, 12] employs online learning methods to find near-optimal prices, rather than directly computing optimal prices for historical data. Adapting such techniques to WAN allocation is a direction for future work.

Incentive issues in routing have also been explored on the supply side, where individual routing nodes may act strategically [13, 15]. We instead consider consumer incentives and assume a single provider coordinates the entire network.

Pretium uses traffic engineering (TE) to provide promises to requests, and to augment its promise, taking link costs into account. Traffic Engineering has been a widely studied problem in networking. Notable works include adaptive congestion avoidance [21], oblivious routing [2, 14], and finding suitable routing parameters for given protocols (e.g., OSPF [14]). Recent papers consider the objective of imposing fairness in a shared network [10, 11].

Traffic engineering for datacenter WANs has been drawn recent attention from both industry and academia [18, 20, 24]. SWAN [18] and B4 [20] aim to improve the utilization of inter-DC WAN. However, the underlying resource management policies of these systems do not compute long-term allocation schedules and do not offer time-guarantees. From the TE perspective, the most relevant work to ours is Tempus [22] which introduces a TE framework which incorporates request demands and deadlines, and allows for guaranteed transfers. However, Tempus does not incorporate (non-linear) link costs into the scheduling framework, and does not consider pricing. To the best of our knowledge, our method of modeling link percentile costs through the sum-of-$k$ proxy (§4.2) is novel.

## 8 Conclusion

In this paper, we present Pretium, a framework that combines dynamic pricing with traffic engineering for inter-datacenter WANs. While Pretium maintains an internal time-varying price for each link in the network, it provides a simple interface to users – each user specifies her request demand and deadline, and can choose a *promise* from a price quote, based on her preferences. A subsequent schedule adjustment module runs periodically to reoptimize flow allocations to manage available capacity and usage costs. Pretium uses the information of historical demands and prices accepted by users to dynamically estimate future demands and update prices. This allows Pretium to quote higher prices during periods of high demand, preferentially serving requests with higher value and shifting traffic with lower value to low utilization periods. Experiments with production WAN traces show that Pretium achieves up to 80% of the welfare of an offline oracular scheme, significantly outperforming various baselines, even when the latter have oracle information on request values and future demands.

## Appendix

**Proof of Theorem 4.2.** Inspired by the bubble sort algorithm, we construct a set of $O(kT)$ constraints and show that the construction results in an upper bound $S_e$ on the sum of the $k$ largest utilization levels in each link. Since we are maximizing $\sum\limits_{i} \sum\limits_{t:t_i^1 \leq t \leq t_i^2} \sum\limits_{r:r \in R_i} X_{irt} \cdot \pi_i - \sum_e C_e S_e$, we are minimizing each $S_e$ and the upper bound becomes tight as required. We will omit subscript $e$ from our notation.We proceed in $k$ iterations: in the first, we "bubble" the largest element, then the second largest, etc.. Our constraints mimic the bubbling operations – for each two numbers $x, y$ to be compared, we have a linear *comparator*, which is manifested through the following inequalities: $x + y = m + M, m \leq x, m \leq y$. Note this implies $M \geq \max\{x, y\}$ and $m \leq \min\{x, y\}$. We note that [25] uses five sorting constraints in their solution; hence, our construction results in 40% less constraints per link, which is substantial especially for large networks. Let $f_j^i$ denote the minimum of the two outputs of the $j$-th comparator at the $i$-th iteration, and let $F_j^i$ denote the maximum of the two values. We use the convention $f_j^0 = f_j$ for all $j \in \{1, 2, \dots T\}$. Accordingly, our first comparator at the first iteration is given by $f_1^0 + f_2^0 = f_1^1 + F_1^1, f_1^1 \leq f_1^0, f_1^1 \leq f_2^0$. As in bubble sort, the maximum output is pushed to the next comparator, i.e., the rest of the constraints for this iteration have following form: $f_j^0 + F_{j-2}^1 = f_{j-1}^1 + F_{j-1}^1, f_{j-1}^1 \leq f_j^0, f_{j-1}^1 \leq F_{j-2}^1$, for every $j \in \{3, 4, \dots T\}$. Using all the above constraints, it can be easily shown that

$$F_{T-1}^1 \geq \max\{f_1^0, f_2^0 \dots\ f_T^0\} \tag{3}$$

$$f_1^0 + f_2^0 + \dots + f_T^0 = f_1^1 + f_2^1 + \dots f_{T-1}^1 + F_{T-1}^1 \tag{4}$$

Proceeding iteratively, we use $(T - i)$ comparators in the $i$-th iteration (all outputs of iteration $i$ excluding $F_{T-i}^i$, are inputs for iteration $i + 1$). Using (4) inductively, we have the following equality after $k$ iterations

$$f_1^0 + \dots + f_T^0 = f_1^k + \dots f_{T-k}^k + F_{T-k}^k + F_{T-k+1}^{k-1} + \dots + F_{T-1}^1. \tag{5}$$

Finally, we add the constraint $S \geq F_{T-k}^k + F_{T-k+1}^{k-1} + \dots + F_{T-1}^1$. Note that we have a total of $O(kT)$ equalities/inequalities.

In order to formally prove that $S$ is not smaller than sum of $k$ largest elements we need the following lemma:

LEMMA 1. *For any $i$ and any set of indices $Y_i \subsetneq \{1, 2 \dots T - i\}$ we can find a subset of indices $Y_{i+1} \subseteq \{1, 2 \dots T - i - 1\}$ such that $|Y_i| = |Y_{i+1}|$ and $\sum_{j \in Y_i} f_j^i \geq \sum_{j' \in Y_{i+1}} f_{j'}^{i+1}$.*

The lemma's proof follows by a charging argument and is omitted here due to lack of space.

We are now ready to prove the theorem. We let $Y_0$ be the set of indices corresponding to the $T - k$ smallest elements among $f_1^0, f_2^0, \dots f_T^0$. And then consequently construct $Y_1, Y_2 \dots Y_k$. We obtain that $Y_k = \{1, 2, \dots T - k\}$. It means that $f_1^k + f_2^k + \dots f_{T-k}^k$ is not larger than the sum of $T - k$-smallest numbers from $f_1^0, f_2^0, \dots f_T^0$. This together with (5) guarantees that $F_{T-k}^k + F_{T-k+1}^{k-1} + \dots + F_{T-1}^1$ is greater or equal to sum of $k$ largest elements from $f_1^0, f_2^0, \dots f_T^0$. □

**Proof of Theorem 5.1.** We will prove that each user request maximizes utility by reporting its request truthfully, if the data being transferred is made unavailable during the period $[t_i^1, t_i^2]$ (this unavailability is the technical assumption alluded to in the statement of Theorem 5.1). It is never beneficial to misreport $S_i$ or $T_i$, nor to report $[\hat{t}_i^1, \hat{t}_i^2] \not\subseteq [t_i^1, t_i^2]$, since the user has no value for data routed incorrectly, early, or late. So assume $[\hat{t}_i^1, \hat{t}_i^2] \subseteq [t_i^1, t_i^2]$ and let $\hat{p}_i(\cdot)$ be the corresponding price menu. We claim that $\hat{p}_i(x) \geq p_i(x)$ for all $x$: this is because $p_i(x)$ has only a larger pool of (route, time) pairs over which to minimize price. This means that reporting $(\hat{t}_i^1, \hat{t}_i^2)$ and requesting $x$ is no better than reporting $(t_i^1, t_i^2)$ and requesting $x$.

Finally, we argue that it is utility-optimal to report only a single request. Breaking a request from $S_i$ to $T_i$ into multi-hop requests (e.g., $S_i$ to $A$, then $A$ to $T_i$) only restricts the set of routes that can be selected, and hence only increases price (as above). Next consider making multiple requests, each from $S_i$ to $T_i$. Since link prices can only increase between requests made in the same timestep, the total payment over all requests can be only greater than the payment of making only a single (truthful) request of the same total size.

## Acknowledgments

We thank Michael Schapira, Mohit Singh, our shepherd Bruce Maggs and the Sigcomm reviewers for their useful feedback.

## 9 References

[1] Gurobi Optimization. http://www.gurobi.com/.

[2] D. Applegate and E. Cohen. Making Intra-Domain Routing Robust to Changing and Uncertain Traffic Demands. In *ACM SIGCOMM*, 2003.

[3] K. J. Arrow and G. Debreu. Existence of an Equilibrium for a Competitive Economy. *Econometrica: Journal of the Econometric Society*, pages 265 – 290, 1954.

[4] B. Awerbuch, Y. Azar, and A. Meyerson. Reducing Truth-telling Online Mechanisms to Online Optimization. In *STOC*, 2003.

[5] M. Babaioff, S. Dughmi, R. Kleinberg, and A. Slivkins. Dynamic Pricing with Limited Supply. In *EC*, 2012.

[6] M. F. Balcan, A. Blum, J. D. Hartline, and Y. Mansour. Reducing Mechanism Design to Algorithm Design via Machine Learning. *J. of Computer and System Sciences*, 74(8):1245 – 1270, 2008.

[7] P. Bangera and S. Gorinsky. Economics of Traffic Attraction by Transit Providers. In *Networking Conference, 2014 IFIP*, pages 1–9, June 2014.

[8] A. Blum, V. Kumar, A. Rudra, and F. Wu. Online Learning in Online Auctions. In *SODA*, 2003.

[9] C. Courcoubetis and R. Weber. *Pricing Communication Networks: Economics, Technology and Modelling*. Wiley Online Library, 2003.

[10] E. Danna, A. Hassidim, H. Kaplan, A. Kumar, Y. Mansour, D. Raz, and M. Segalov. Upward Max Min Fairness. In *INFOCOM*, 2012.

[11] E. Danna, S. Mandal, and A. Singh. A Practical Algorithm for Balancing the Max-Min Fairness and Throughput Objectives in Traffic Engineering. In *INFOCOM*, 2012.

[12] N. R. Devanur and T. P. Hayes. The Adwords Problem: Online Keyword Matching with Budgeted Bidders Under Random Permutations. In *EC*, 2009.

[13] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A BGP-based Mechanism for Lowest-cost Routing. *Distributed Computing*, 18(1):61–72, 2005.

[14] B. Fortz and M. Thorup. Internet Traffic Engineering by Optimizing OSPF Weights in a Changing World. In *INFOCOM*, 2000.

[15] P. B. Godfrey, M. Schapira, A. Zohar, and S. Shenker. Incentive Compatibility and Dynamics of Congestion Control. In *ACM SIGMETRICS*, 2010.

[16] S. Ha, S. Sen, C. Joe-Wong, Y. Im, and M. Chiang. TUBE: Time-dependent Pricing for Mobile Data. In *ACM SIGCOMM*, 2012.

[17] M. T. Hajiaghayi, R. Kleinberg, M. Mahdian, and D. C. Parkes. Online Auctions with Re-usable Goods. In *EC*, 2005.

[18] C. Y. Hong et al. Achieving High Utilization with Software-Driven WAN. In *ACM SIGCOMM*, 2013.

[19] J. Hsu, J. Morgenstern, R. M. Rogers, A. Roth, and R. Vohra. Do prices coordinate markets? *CoRR*, abs/1511.00925, 2015.

[20] S. Jain et al. B4: Experience with a Globally-Deployed Software Defined WAN. In *ACM SIGCOMM*, 2013.

[21] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the Tightrope: Responsive Yet Stable Traffic Engineering. In *ACM SIGCOMM*, 2005.

[22] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula. Calendaring for Wide Area Networks. In *ACM SIGCOMM*, 2014.

[23] F. Kelly, A. Maulloo, and D. Tan. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability. In *Journal of the Operational Research Society*, 1998.

[24] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter Bulk Transfers with Netstitcher. In *ACM SIGCOMM*, 2011.

[25] H. H. Liu et al. Traffic Engineering with Forward Fault Correction. In *ACM SIGCOMM*, 2014.

[26] S. Low and D. Lapsley. Optimization Flow Control, I: Basic Algorithm and Convergence. In *IEEE/ACM Transactions on Networking*, Dec 1999.

[27] R. T. B. Ma, D. M. Chiu, J. C. S. Lui, V. Misra, and D. Rubenstein. Internet Economics: The Use of Shapley Value for ISP Settlement. In *IEEE/ACM Transactions on Networking*, June 2010.

[28] R. G. Michael and S. J. David. Computers and Intractability: A Guide to the Theory of NP-completeness. *W.H Freeman*, 1979.

[29] C. Raiciu et al. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *NSDI*, 2012.

[30] S. Shenker, D. Clark, D. Estrin, and S. Herzog. Pricing in Computer Networks: Reshaping the Research Agenda. *ACM SIGCOMM CCR*, Apr. 1996.

[31] V. Valancius, N. Feamster, R. Johari, and V. Vazirani. MINT: A Market for INternet Transit. In *ACM CONEXT*, 2008.

[32] V. Valancius, C. Lumezanu, N. Feamster, R. Johari, and V. V. Vazirani. How Many Tiers?: Pricing in the Internet Transit Market. In *ACM SIGCOMM*, 2011.

[33] H. Varian. *Microeconomic Analysis*. Norton International edition. W.W. Norton, 1992.

[34] H. Zhang et al. Guaranteeing Deadlines for Inter-datacenter Transfers. In *Eurosys*, 2015.

[35] L. Zhang, W. Wu, and D. Wang. The Effectiveness of Time Dependent Pricing in Controlling Usage Incentives in Wireless Data Network. In *ACM SIGCOMM*, 2013.