FCP: A Flexible Transport Framework for Accommodating Diversity

Dongsu Han Robert Grandl† Aditya Akella† Srinivasan Seshan*
KAIST University of Wisconsin-Madison† Carnegie Mellon University*
{dongsuh,srini}@cs.cmu.edu, {rgrandl,akella}@cs.wisc.edu

ABSTRACT

Transport protocols must accommodate diverse application and network requirements. As a result, TCP has evolved over time with new congestion control algorithms such as support for generalized AIMD, background flows, and multipath. On the other hand, explicit congestion control algorithms have been shown to be more efficient. However, they are inherently more rigid because they rely on in-network components. Therefore, it is not clear whether they can be made flexible enough to support diverse application requirements. This paper presents a flexible framework for network resource allocation, called FCP, that accommodates diversity by exposing a simple abstraction for resource allocation. FCP incorporates novel primitives for end-point flexibility (*aggregation* and *preloading*) into a single framework and makes economics-based congestion control practical by explicitly handling load variations and by decoupling it from actual billing. We show that FCP allows evolution by accommodating diversity and ensuring coexistence, while being as efficient as existing explicit congestion control algorithms.

Categories and Subject Descriptors

C.2.2 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Protocols; C.2.5 [**COMPUTER-COMMUNICATION NETWORKS**]: Local and Wide-Area Networks—*Internet*

Keywords

Transport protocol; congestion control; end-point flexibility

1. INTRODUCTION

Networked applications require a wide range of communication features, such as reliability, flow control, and in-order delivery to operate effectively. Since the Internet provides only a very simple, best-effort, datagram-based communication interface, we have relied on transport protocols to play the key role of implementing the application's desired functionality. As application needs and workloads have changed and diversified, transport protocols have adapted to provided diverse functionality to meet their needs. In general, changes to transport protocols, such as adding better loss recovery mechanisms, have been relatively simple since they only require unilateral changes at the endpoints.

However, among the functions that transport protocols implement, congestion control is unique since it concerns resource allocation, which requires coordination among all participants to ensure high link utilization and fairness. For example, two very different styles of congestion control (e.g., TCP and RCP [16] or D^3 [49]) cannot coexist as they interfere with each other's resource allocation. Nevertheless, today's applications impose increasingly diverse requirements for resource allocation such as utilizing multipath [50], supporting deadlines [49], and optimizing for non-traditional metrics including quality of user experience [3].

The key requirements for supporting diversity in congestion control are 1) the flexibility to handle new requirements and 2) ensuring coexistence between different behaviors without negative impact on efficiency and fairness. Fortunately, TCP-style congestion control has been able to support a wide range of congestion control techniques to meet different application requirements; these include support for: streaming applications that require bandwidth guarantees [7, 17] or low-latency recovery [23, 32], non-interactive applications that can leverage low-priority, background transfer [48], applications that require multi-path communication [50] for robustness, and Bittorrent-like content distribution that transfers objects from multiple sources.

Two key aspects of TCP's congestion control enable diversity: 1) its purely end-point based nature enables each end-point the *flexibility* to employ different algorithms and 2) the notion of TCP-friendliness [17] provides a mechanism for *coexistence* between different algorithms and behaviors.

However, router-assisted explicit congestion control, such as RCP and XCP, is far more efficient than TCP, achieving high utilization, small delays, and faster flow completion time [16, 25, 49]. Recently, D^3, a customized explicit congestion control algorithm for datacenters, has been shown to significantly outperform TCP [49]. On the other hand, router-assisted congestion control algorithms are far less flexible because the network's feedback strictly defines end-point behaviors. In fact, the two properties of TCP that enable diversity do not hold in router-assisted congestion control.

Since router-assisted congestion control does not provide flexibility and end-point based algorithms do not provide high efficiency, achieving high efficiency and supporting diversity appear to be at odds. Contrary to this belief, this paper shows that we can have the best of both worlds. Herein, we present the design of FCP (Flexible Control Protocol), a novel congestion control framework that is as efficient as explicit congestion control algorithms (e.g., RCP and XCP), but retains (or even expands) the flexibility of an end-point based congestion control.

SIGCOMMâĂŹ13, August 12–16, 2013, Hong Kong, China.

Our primary contribution is showing that *explicit congestion control algorithms can be made as flexible as end-point based algorithms*. To achieve this, FCP incorporates two key ideas:

1. To maximize end-point flexibility without sacrificing efficiency, FCP effectively combines fine-grained end-point control and explicit router feedback. To our knowledge, FCP is the first congestion control algorithm that incorporates the two.
2. To ensure safe coexistence of different behaviors within the same network, FCP introduces a simple invariant for fairness.

To enable flexible resource allocation, we leverage and extend ideas from economic-based congestion control [26, 28]. In particular, we allow each domain to allocate resources (budget) to a host and make networks explicitly signal the congestion price. Such *aggregation* enables end-points to freely assign their own resources to their flows and allows the network to assign different pricing schemes to different classes of flows for extra functionality. To ensure safe coexistence of different end-point resource allocation strategies, the system maintains a key invariant that the amount of traffic a sender can generate is limited by its budget, which is the maximum amount it can spend per unit time.

However, numerous practical challenges exist in generating explicit price feedback, while allowing the end-points to dynamically change their flows' budget allocation. In particular, existing explicit congestion control algorithms exhibit undesirable behaviors during transient periods. For example, XCP results in unfair bandwidth allocation [16], and RCP can significantly overload the network under rapid variations in load. Furthermore, the notion of a flow does not fit well with the request/response-like, short, bursty workloads of many applications.

To design a more practical and robust system, FCP further improves upon past designs in two critical ways:

1. Unlike other router-assisted designs [16, 25, 28], FCP accommodates high variability and rapid shifts in workload, which is critical for the flexibility and performance. This property comes from our *preloading* feature that allows senders to commit the amount of resources they want to spend ahead of time. Preloading generalizes the step-change algorithm in [27], making FCP end-points a more active component in congestion control that explicitly handles short, bursty workloads.
2. We address practical issues in system design and resource management, such as dealing with relative pricing. FCP provides a practical mechanism for cost-fairness [10] that is decoupled from actual billing, unlike other approaches [10, 26], while enforcing that each sender operates within its own budget.

Our in-depth evaluation based on simulations and real implementation shows that FCP is locally stable, achieves faster and graceful convergence, and outperforms existing explicit congestion control algorithms in many cases, without imposing unreasonable overhead. FCP's end-point flexibility and coexistence mechanisms allow end-points to easily implement diverse strategies, benefiting many types of applications. Furthermore, the pricing abstraction coupled with preloading allows FCP to add support for new service models and various quality-of-service features.

In the rest of the paper, we discuss our motivation and related work (§2), present our design (§3), and discuss practical issues in deployment (§4). We then evaluate FCP (§5), describe budget management issues (§6), and conclude in §7.

2. MOTIVATION AND RELATED WORK

This section presents the key requirements (§2.1), related work (§2.2) and principles of FCP design (§2.3).

2.1 Diversity and Flexibility

The congestion control algorithm must accommodate *diversity* such that different strategies can coexist and permit *flexibility* to ensure that new behaviors can be implemented to accommodate potential changes in communication patterns.

Diversity: To understand the nature of diversity in resource allocation, we categorize previous work into four categories:

1. Local resource allocation: Prior work, such as congestion manager [5], SCTP [37], and SST [19], has shown the benefits of sharing bandwidth across (sub-) flows that share a common path.
2. Allocating resources within a network: Support for differential or weighted bandwidth allocation across different flows has been explored in generalized AIMD [6, 51], MulTCP [13, 14], and TCP Nice [48].
3. Network-wide resource allocation: Bandwidth is sometimes allocated on aggregate flows or sub-flows. Multipath TCP (MPTCP) [50] controls the total amount of bandwidth allocated to its sub-flows. Distributed rate limiting [41] and assured forwarding in DiffServ control resource allocation to a group of flows. In such systems, flows that do not traverse the same path are allocated shared resources (e.g., MPTCP is fair to regular TCP at shared bottlenecks). Thus, one can be more aggressive in some parts of the network at the expense of being less aggressive in other parts [41].
4. Bandwidth stability: Although strict performance guarantees require in-network support, transport protocols try to provide some level of performance guarantees. TFRC [17] provides slowly changing rates, and OverQoS [46] provides probabilistic performance guarantees.

Flexibility: Congestion control must be flexible enough to introduce new behaviors to handle new requirements. TCP's flexibility enables new algorithms and implementations to be introduced over time at the end-points as well as active queue management in the network. This is increasingly important because future networks might support new types of service models [24], and application developers desire to optimize for new application-specific metrics, such as the fraction of flows meeting a deadline [49] and quality of user experience [3]. To ensure flexibility, FCP must expose sufficient control to the end-host as well as to the network. In addition, a scalable mechanism is needed in order to exert control over aggregate flows without per-flow state at the router.

Note that apart from these requirements for evolution, there are also traditional requirements, such as high efficiency, fast convergence, and fair bandwidth allocation. TCP Cubic [22], XCP [25], CSFQ [45], and many others [16, 18] fall into this category. Our goal is to design a flexible congestion control framework that accommodates diversity and allows flexibility in resource allocation, while meeting these traditional requirements.

2.2 Related Work

Our framework builds upon concepts from many previous designs to provide a more general framework for resource allocation.

Economics-driven resource allocation has been explored from the early days of the Internet [12, 38, 42]. MacKie-Mason and Varian [35] proposed a *smart market* approach for allocating bandwidth. Since then many have tried to combine this approach with congestion control, which led to the development of economics-based congestion control [11, 28, 34] and network utility maximization (NUM) [52].

Kelly [28] proved that in a system where users choose to pay the charge per unit time, the transfer rate could be determined such that the total utility of the system is maximized at equilibrium, and the rate assignment satisfies the weighted proportional fairness criterion. Two classes of algorithms can achieve such rate allocation [28]:

1) In algorithms that correspond to the primal form of the utility maximization problem, the network implicitly signals the congestion price, and the senders use additive increase/multiplicative decrease rules to control rate. 2) In the dual algorithm, the network uses explicit rate feedback based on shadow prices.

The primal algorithms [2, 11, 13, 30, 33, 36] use end-point based congestion control. Gibbens and Kelly [21] show how marking packets (with an ECN bit) and charging the user the amount proportional to the number of marked packets achieves the same goal, and allows evolution of end-point based congestion control algorithms. Re-ECN [11] shows how end-users can avoid variable and unpredictable congestion charging by purchasing fixed congestion quotas, while using economics-based congestion control on top of TCP. Many congestion control and active queue management schemes [30, 33, 36], such as REM [2], have also been developed to achieve Kelly's (weighted) proportional fairness. However, they abandon the notion of actual congestion charging, but resort to (weighted) flow rate fairness. The dual algorithm uses rate (or price) as an explicit feedback and was materialized by Kelly et al. [27] by extending RCP [16][1].

However, the flow rate fairness used by the above algorithms is far from ideal [10, 11, 40]. Just as in TCP, one can obtain more bandwidth just by initiating more flows, a concern in many environments including recent cloud computing [40]. Similar to re-ECN [11], FCP departs from flow rate fairness, but assigns the resources to a group of flows (e.g., flows that originate from a single host) and achieves weighted proportional fairness. However, unlike re-ECN, where users have to purchase a congestion quota, FCP completely decouples the real-world billing from congestion pricing and handles practical problems that arise from it. Furthermore, FCP's preloading allows the end-point to play an active role in congestion control, and naturally supports bursty workloads, flow departure and arrival (§5.1), flexibility at the end-point in resource allocation (§5.3), and quality of service features in the network (§5.4).

Extensible transport: Others focus on designing an extensible [9, 39] or configurable [8] transport protocol, either focusing on security aspects of installing mobile code at the end-host or taking software engineering approaches to modularize transport functionality at compile time. However, the core mechanism for coexistence is TCP-friendliness.

Virtualization: Finally, virtualization [1, 43] partitions a physical network, allowing completely different congestion control algorithms to operate within each virtualized network. However, slicing bandwidth creates fragments and reduces the degree of statistical multiplexing, which we rely on for resource provisioning. Also, this increases the complexity of applications and end-hosts that want to use multiple protocols simultaneously as they have to participate in multiple slices and maintain multiple networking stacks.

2.3 Principles of Design

FCP employs two key principles for accommodating diversity and ensuring flexibility: *aggregation* and *local control*. We explain them in detail and show that it requires a careful feedback design in order to achieve both properties.

Aggregation: FCP assigns resources to a group of flows. Aggregation allows the network to control the amount of resources that the group is consuming in a distributed and scalable fashion while preserving the relative weight of the individual flows within the group. FCP ensures that, regardless of the strategy that an end-point might use, its aggregate resource consumption is proportional to its budget. This is critical for the coexistence of different end-point behaviors as it provides fairness. Aggregation also simplifies control and enforcement; the network can enforce the constraint that a host generates traffic within its allocated resources without the routers having to maintain per-flow state. For additional flexibility, we allow aggregation at various levels. ISPs can also aggregate traffic coming from another ISP and apportion their own resources (§6). For example, each domain can independently assign weights to neighboring domains' traffic or to a group of flows.

Local control: In FCP, both the end-host and the network have local control; the network controls the allocation of bandwidth on aggregate flows, and end-hosts decide how to use their own resources (budget) given the network constraint. However, explicit congestion control algorithms, such as XCP and RCP, leaves no local control at the end-point because their rate feedback strictly defines the end-point's behavior. In contrast, FCP's local control gives end-points control and freedom in using their own resources, making them an active component. In FCP, it is the responsibility of the end-point to decide how it distributes its own resources locally amongst its own flows (diversity category 1 in §2.1). The network then assigns bandwidth proportional to the flow's assigned resources (category 2). A host can also spend more resources on some flows while spending less on others (category 3). Various entities in the network may also control how their own resources are shared between groups of flows. For example, networks can allocate bandwidth in a more stable manner to certain groups of flows (category 4).

Designing the form of network-to-end-point feedback that meets both of these requirements is challenging. Providing explicit rate feedback leaves little flexibility at the end-point; however, using an abstract or implicit feedback mechanism, such as loss rate or latency, or loosely defining the semantics of feedback, allows a broader range of end-host behaviors. However, using such feedback typically involves guesswork. As a result, end-hosts must probe for bandwidth, sacrificing performance and increasing the convergence time (e.g., in a large bandwidth-delay product link). Providing differential feedback for differential bandwidth allocation is also hard in this context. As a result, differential bandwidth allocation typically is done by carefully controlling how aggressively end-points respond to feedback relative to each other [51]. This also makes enforcement and resource allocation on aggregate flows very hard. For example, for enforcement, the network has to independently measure the implicit feedback that an end-point is receiving and correlate it with the end-point's sending rate.

We take a different approach by leveraging ideas from economics-based congestion control [21, 28]. Using *pricing* as a form of explicit feedback, we design a flexible explicit congestion control algorithm that supports aggregation and local control.

3. DESIGN

We now sketch our high-level design in steps.

(1) Each sender (host) is assigned a budget ($\$/sec$), the maximum amount it can spend per unit time. The budget defines the weight of a sender, but is different from the notion of "willingness to pay" in [28] in that it does not have a real monetary value. We first focus on how FCP works when the budget is assigned by a centralized entity. Later, in §6, we extend FCP to support distributed budget management in which each domain assigns budgets completely independently and may not trust each other.

(2) At the start of a flow, a sender allocates part of its budget to the flow. This per-flow budget determines the weight of the flow. The budget controller in the operating system *dynamically* assigns budget to flows taking into account traffic demand and application objectives. The end-point flexibility is enabled at this stage because

[1]Prior to this, Low and Lapsley's preliminary design [33] also incorporates explicit price feedback.

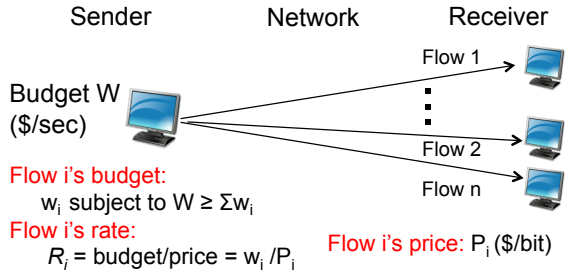


Figure 1: Design Overview

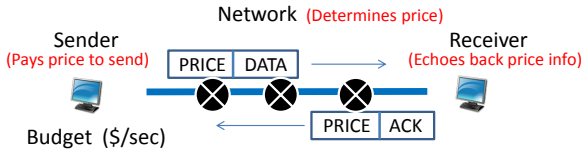


Figure 2: Network generates the price feedback.

the sender can implement various strategies of resource allocation, as illustrated in §3.2.

Figure 1 illustrates the budget assignment across flows. The sender has a budget of W, and can distribute its budget to its flows as it pleases provided that the sum of the flows' budgets, $\sum w_i$, is less than or equal to W. The rate of flow i, R_i, is then defined as w_i/P_i where P_i is the price for the path flow i is traversing. This allows the end-host to control its own resources and achieve differential bandwidth allocation on a flow-by-flow basis. Next, we show how the path price is generated.

(3) The network determines the congestion price ($\$/bit$) of each link. In FCP, the sender learns the path price in the form of explicit feedback. Similar to [28], the price of path r, P_r, is defined as the sum of link prices: $P_r = \sum_{l \in r} p_l$, where p_l is the price of link l in path r. To ensure efficiency, the price adapts to the amount of budget (traffic times its price) flowing through the system. In §3.1, we first show a uniform pricing scheme in which all packets see the same price. Later in §3.2, we show how networks can employ differential pricing and aggregate congestion control to support various features such as quality of service and multicast congestion control.

(4) The flow must ramp up to its fair-share, R_i, by actually using its budget w_i. Two problems arise at this stage: At the start of the flow the sender does not know the path price, P_i. More importantly, the network does not know the budget amount (w_i) it should expect. This leads to serious problems especially when the load changes rapidly. For example, when a path is not congested, its congestion price will be an infinitesimal value, ε. Any $w_i >> \varepsilon$ will set the sending rate to an infinite value, and overload the network significantly. To address this problem, we introduce *preloading*, a distinct feature of our design that allows a host to rapidly increase or decrease a flow's budget. Preloading allows a sender to specify the amount of budget increase for the next round in multiples of the current price, allowing the network to adjust the expected input budget and generate the price based on this. The challenge here is to make hosts and routers update the budget w_i and the price P_i in a coordinated fashion so that the host generates traffic up to its budget only when the network is expecting this. This is especially challenging because the system is asynchronous and feedback is delayed. To achieve this in FCP, routers update the price on every packet reception, and hosts preload on a packet-by-packet basis when the flow initially ramps up or the budget assignment changes.

The sender ramps up in two steps using preloading:

(4-a) In the first packet, we specify (preload) how much we want to send in the next RTT. However, at this point we do not yet know the path price. Therefore, we preload a conservative amount. (See §3.1 for details.)

(4-b) After the price is discovered, the sender can preload the appropriate amount for using up the budget assigned to the flow. Preloading is also used when an active flow's budget assignment changes.

3.1 Flexible Control Framework

We now describe the algorithm in detail. In FCP, the system maintains the following *invariant*: For all hosts h,

$$\sum_{s \in Packets} price(s) \cdot size(s) \leq W_h \quad (1)$$

where *Packets* is the set of packets sent by host h during unit time, price(s) is the most recent price of the path that packet s is sent through, and W_h is host h's budget.

Explicit price feedback: To satisfy the invariant, senders have to know the path price. As Figure 2 illustrates, an FCP header contains a price value, which accumulates the path price in the forward path and gets echoed back to the sender. Each router on the path updates the price in the header as: $price = price + p_l$, where p_l is the egress link price.

Pricing ensures network efficiency by dynamically adapting the price to the amount of incoming budget. We first show how each router calculates the link price without considering the preloading. Each link's price must reflect the amount of incoming budget and the capacity of the link. Upon receiving packet s at time t, each router calculates the link price per bit $p(t)$ ($\$/bit$) as:

$$p(t) = \frac{I(t)}{C - \alpha q(t)/d} \quad (2)$$

$$I(t) = \frac{\sum_{s \in (t-d,t]} p(t - rtt(s)) \cdot size(s)}{d} \quad (3)$$

Equation 2 sets the price as the incoming budget (amount of traffic times its price) over remaining link capacity. The numerator, $I(t)$, denotes the total incoming budget per unit time ($\$/sec$), which is the sum of all packets' prices ($) seen during the averaging window interval (t-d,t]. The denominator reflects the remaining link capacity, where C is the link-capacity, $q(t)$ is the instantaneous queue size, α is a constant, and d is the averaging window. $rtt(s)$ is the RTT of the packet s, and $p(t - rtt(s))$ is the past feedback price per bit. Unlike other explicit congestion control protocols [16, 25], the router calculates the link price at every packet reception, and keeps the time series $p(\cdot)$. We set d as multiples of average RTT (2 to 3 times the average RTT in our implementation).

Equation 2 deals with efficiency control by quickly adapting the price based on the incoming budget and the remaining link capacity. Fairness is achieved because everyone sees the same price. Therefore the bandwidth allocation is proportional to budget. In §5.1, we demonstrate the local stability of this algorithm by testing it under random perturbations.

Equation 3 estimates the amount of incoming budget that a router is going to see in the future using recent history. When the incoming budget, $I(t)$, is relatively constant over time the price is stable. However, when the input budget constantly changes by a large amount (e.g., due to flow departures and arrivals or budget assignment changes), the price will also fluctuate. This, coupled with the inherent delayed feedback, can leave the system in an undesirable state for an extended period. During convergence, the network may see high loss rate, under-utilization, or unfairness, a

common problem for explicit congestion control algorithms [16] as we show in §5.1.

This problem has been regarded as being acceptable in other systems because changes in load are viewed as either temporary or incremental [31]. However, this is not the case in our system. One of the key enablers of evolution in FCP is the end-host's ability to arbitrarily assign its budget to individual flows, and we expect that rapid change in the input budget will be the norm. Increasing budget rapidly also allows senders to quickly ramp up their sending rates to their fair-share. However, this is especially problematic when the amount of budget can vary by large amounts between flows. For example, consider a scenario where a link has a flow whose budget is $1\$/sec$. When the flow changes the budget assignment to 1000 $\$/sec$ instantly, this link will see 1000x the current load. However, preventing this behavior and forcing users to incrementally introduce budget will make convergence significantly slower and limit the flexibility.

Preloading allows a host to rapidly increase or decrease the budget amount per flow on a packet-by-packet basis without such problems. It provides a mechanism for the sender to specify the amount of budget it intends to introduce in the next round, allowing the network to adjust the expected input budget and generate prices based on the commitment.

However, the sender cannot just specify the absolute budget amount that it wants to use because the path price is a sum of link prices and routers do not keep track of the path price, but only the local link's price. Instead, we let senders preload in multiples of the current price. For example, if the sender wants to introduce 10 times more budget in the next round, it specifies the preload value of 10. For routers to take this preload into account, we update Equation 3 as follows:

$$I(t) = \frac{\sum_{s \in (t-d,t]} p(\cdot)\, size(s)\, (1 + preload(s) \cdot d/rtt(s))}{d} \quad (4)$$

The additional preload term takes account for the expected increase in the incoming budget, and $rtt(s)$ accounts for the difference between the flow's RTT and the averaging window. Preloading provides a hint for routers to accurately account for rapid changes in the input budget. Using the preload feature, FCP handles flow arrival, departure, and short flows explicitly, and thus significantly speeds up convergence and reduces estimation errors.

Header: We now describe the full header format. An FCP data packet contains the following congestion header:

RTT	price	preload	balance

When the sender initializes the header, RTT field is set to the current RTT of the flow, price is set to 0, and preload to the desired value (refer to sender behavior below). The balance field is set as the last feedback price—i.e., the price that the sender is paying to send the packet. This is used for congestion control enforcement (§4). Price and preload value are echoed back by an acknowledgement.

Sender behavior with preloading: Senders can adjust the allocation of budget to their flows at any time. Let x_i be the new target budget of flow i, and w_i the current budget whose unit is $\$/sec$. When sending a packet, it preloads by $(x_i - w_i)/w_i$, the relative difference in budget. When an ACK for data packet s is received, both the current budget and the sending rate are updated according to the feedback. When preload value is non-zero, the current budget is updated as:

$$w_i = w_i + paid \cdot size(s) \cdot preload/rtt$$

where *paid* is the price of packet p in the previous RTT. The sending rate r_i is updated as: $r_i = w_i/p_i$, where p_i is the price feedback in the ACK packet. Note that preloading occurs on a packet-by-packet basis and the sender only updates the budget after it receives the new price that accounts for its new budget commitment. Also note that the preload can be negative. For example, a flow preloads -1 when it is terminating. Negative preloading allows the network to quickly respond to decreasing budget influx, and is useful when there are many flows that arrive and leave. Without the negative preload, it takes an average window (d) for the budget of an already departed flow to completely decay in the system.

Start-up behavior: We now describe how FCP works from the start of a flow. We assume a TCP-like 3-way handshake. Along with a SYN packet, we preload how much we want to send in the next RTT. However, because we do not yet know the path price, we do not aggressively preload. In our implementation of FCP, we adjust the preload so that we can send 10 packets per RTT after receiving a SYN/ACK. The SYN/ACK contains the price. Upon receiving it, we initialize the budget assigned to this flow w_i as: $price \cdot size \cdot preload/rtt$. The sender then adjusts its flows' budget as described earlier.

3.2 Supporting Diversity

First, we illustrate the type of flexibility FCP enables at the end-points. FCP's local control and aggregation allow flexibility similar to that of end-point based algorithms in that they give end-points significant control over how the resources are allocated. Furthermore, preloading enables dynamic resource allocation at the packet-level. Next, we show that FCP also provides significant flexibility in the network with differential feedback.

End-point flexibility: FCP allows end-points to focus on their own optimization objectives and resource allocation strategies without being concerned about the network-wide objectives, such as efficiency and fairness. It truly decouples the two, enabling end-points to employ intelligent strategies and improve their implementation over time. Below, we outline several strategies as examples.

- **Equal budget** achieves flow rate fairness within a single host by equally partitioning the budget between its flows (e.g., when there are n flows, each flow gets $budget/n$). The throughput of flows within a single host will be purely determined by the path's congestion level.
- **Equal throughput**: An end-point may want to achieve equal sending rates to all communicating parties (e.g., in a conference call). This can be achieved by carefully assigning more budget to more expensive flows.
- **Max throughput** tries to maximize the total throughput given the total budget by allocating more budget towards inexpensive paths (less congested) and using less on congested paths. This generalizes the approach used in multipath TCP [50] even to flows sent to different destinations.
- FCP also supports **background flows** [48] that only utilize "spare network capacity". When foreground flows take up all the capacity, background flows must transmit at a minimal rate. This can be achieved by assigning a minimal budget to background flows. When links are not fully utilized, the price goes down to "zero" and path price becomes marginal. Therefore, with only a marginal budget, background flows can fill up the capacity.
- **Statistical bandwidth stability**: Flows that require a stable throughput [17] can be supported by reallocating budget between flows; if the flow's path price increases (decreases), we increase (decrease) its budget. When the budget needs to increase, the flow steals budget from normal flows. This is slightly different from the smooth bandwidth allocation of TFRC [17] in that temporary variation is allowed, but the average throughput over a few RTTs is probabilistically stable. The probability depends on how much

budget can be stolen and the degree of path price variation. Such a statistical guarantee is similar to that of OverQoS [46].

Note this list is not exhaustive, and FCP's flexibility allows end-hosts to optimize a variety of metrics. Servers in CDNs may employ a resource allocation strategy to maximize the quality of user experience [3], which affects users engagement [15]. Furthermore, these algorithms are not mutually exclusive. Different hosts or different groups of flows within a single host can use different strategies. Coexistence is guaranteed because end-hosts stick to the fairness invariant of Equation 1. Thus, FCP facilitates the design and deployment of heterogeneous behaviors in resource allocation.

Network and end-point flexibility: End-points and networks can also simultaneously evolve to achieve a common goal. FCP's local control and aggregation allow routers to give differential pricing. Preloading also allows dynamic resource allocation. We show how one might use them to provide extra functionality. In our examples below, the algorithm changes from FCP's original design, including the path price calculation, link price generation, and preloading strategy. However, the invariant of Equation 1 remains unchanged.

- FCP can support a stricter form of **bandwidth stability** with differential pricing and end-point cooperation by bounding the price variability for bandwidth stability flows. For stability flows, end-points can only double the flow's budget which limits the speed at which the rate can ramp up. Routers have two queues per link: stability and normal. The stability queue's price variation is bounded by a factor of two, during the time window d (e.g., twice the average RTT). When the price has to go up more than that amount, it steals bandwidth from the normal queue to bound the price increase of the stability queue. As a result, the normal queue's price increase even more, but the bandwidth stability queue's price is bounded. When the price might go down by a factor of two during a window, it assigns less bandwidth to the stability queue. At steady state, the prices of both queues converge to the same value.
- FCP's flexibility allows the network to support a different service model. For example, FCP can support **multicast congestion control**. The price of a multicast packet is the sum of the price of all links that it traverses. We sum up the price of a packet in the following recursive manner to account for each link price only once in a multicast tree: When a router receives a multicast packet, it remembers its upstream link price, resets the price feedback to zero, and sends out a copy of the packet to each of its outgoing interfaces in the multicast tree. Receivers echo back the price information. Upon receiving an ACK, each router remembers the price of its subtrees and sends a new ACK containing a new price for its upstream router. This new price is the sum of all the price feedback from its subtrees and the uplink price that it previously remembered.
- FCP can also support Paris Metro Pricing (PMP)-like [38] differential pricing. The network can assign higher price to a priority service in multiples of the standard price. For example, the priority queue's price can be 10 times the standard queue. The end-host will only use the priority queue when it is worth it and necessary, making the priority queue see much less traffic that the standard one.
- FCP can implement D^3[49]-style **deadline support** using differential pricing with two queues per link: a deadline queue and a best effort queue. The deadline queue always gives a small agreed-upon fixed price. Flows indicate their desired rate by preloading. The router gives the fixed price only when the desired rate can be satisfied. Otherwise, it puts the packet into the best-effort queue and gives a normal price to match the aggregate rate to the normal queue's bandwidth. A deadline flow first tries deadline support, but falls back to best effort service when it receives a price different from the fixed value after preloading (See §5.4 for details).

4. PRACTICAL ISSUES

Section 3 addressed the first order design issues. We now address remaining issues in implementation and deployment.

Minimum price: In economics-based congestion control [27], the price per unit charge goes to zero when the willingness to pay, w, is an infinitesimal value or the link is constantly under-utilized. In RCP, this means that the rate feedback can be an infinitely large value. When this happens, the convergence behavior of a new flow is far from ideal. [2] To address this problem, we define a globally agreed upon minimum price and treat this value as zero (e.g., for any price P, $P = P + MININUM$). Therefore, an uncongested path has the minimum price. However, defining a minimum price has two implications in budget assignment: 1) For a host with a unit budget to saturate a path, this minimum price must be smaller than unit budget over the maximum link capacity. 2) To differentiate background flows from normal flows, normal flows must have far more budget. Thus, we make the minimum price sufficiently low ($10^{-18}\$/byte$), ensuring a flow to saturate $\$1/Exabyte$ given a unit budget, while giving a sufficient range for background flows.

Enforcement: FCP's explicit price feedback allows the network to perform enforcement without per-flow state (but with per-user state at the edge router). We leave the exact mechanisms as future work, but briefly describe the high-level approach. The network can enforce 1) whether the sender is paying the right amount, 2) whether the sender is operating within its budget, and 3) the sender is not misbehaving. To enforce 1), we use the enforcement mechanism in Re-feedback [11]. The sender sets the `balance` field to the amount it is paying, and every router along the path subtracts the link price for this packet, $p(t - rtt)$. If the amount reaches a negative value, this means that the sender did not pay enough. The egress edge router maintains an average of the balance value. If the value is consistently below zero, the egress router computes the average balance for each sender and identifies who is consistently paying less, and drops its packet. This information is then propagated upstream to drop packets near the source. Details on statistical methods for efficient detection are described in [11]. To enforce 2), the ingress edge router enforces a modified version of invariant that accounts for preloading:

$$\sum_{\text{packet in (t-1,t]}}^{\text{for each}} balance(packet) \cdot size \cdot preload \leq budget$$

This requires a classifier and per-host state at the ingress edge, and ensures that a host does not increase its budget arbitrarily. Ingress edge routers can drop packets once a host uses more budget than it is assigned. Additionally, the network can enforce the preloading behavior. A misbehaving flow may not preload and generate lots of traffic. To enforce 3), the edge router can ensure that senders cannot overuse the amount that it has previously committed. Later in §6, we discuss how our inter-domain budget management can also mitigate the problem of misbehaving users across domains.

Incremental deployment: Supporting FCP flows to traverse a partially FCP-enabled path requires more research. Our focus

[2] RCP's implementation avoids the problem by setting the maximum rate feedback to be the link price. However, this solution is not applicable to weighted proportional-fairness [27] because the rate feedback (inverse of price per unit charge) can be larger than the capacity at equilibrium.

instead is on deploying FCP locally while ensuring backwards-compatibility; e.g., allowing legacy TCP traffic to traverse an FCP-enabled segment. It is worthwhile to consider local deployments, similar to those in D2TCP [47] and D3 [49], as they provide a path towards wider adoption.

End-points only use FCP when the entire path is FCP-enabled. When only a part of the end-to-end path is FCP enabled, end-hosts use TCP, but the FCP routers perform FCP-style congestion control on aggregate TCP flows within the FCP segment. When FCP routers receive TCP traffic, they maintain a single separate queue for TCP. The routers then map TCP traffic to a new FCP flow by attaching an FCP header and allocating some budget. The budget amount can be either fixed or relative to other concurrent FCP flows. The price information is periodically sent back to the upstream FCP router. The new FCP flow's sending rate then determines aggregate TCP throughput. TCP packets are dropped when the TCP queue is full, triggering TCP's congestion control.

Computational overhead and optimizations: To calculate the price, routers need to keep a moving window of input budget and update the average RTT. When packets arrive at the router, it updates the statistics and assigns a timer for the packet so that its value can expire when it goes out of the window. It requires roughly 20 floating point operations for each packet. We believe high-speed implementation is possible even with software implementations on modern hardware.[3]

We perform two optimizations in our implementation for greater robustness and lower overhead: 1) We make our implementation robust to RTT measurement errors. Equation 4 uses past price, $p(t - rtt(s))$, but when price variation is large around time $t - rtt(s)$, small measurement error can adversely impact router's estimate of the actual past price. Even worse, routers may not store the full price history. We take the minimum of $p(\cdot)$ and *balance* field in the congestion header. This bounds the error between the paid price of the sender and the router's estimation, even when a router does not fully remember its past pricing. 2) Our implementation keeps a summary of the price history to optimize for performance. By default, it keeps a price history up to 500 ms at every 10 usec interval. The router updates the recent price history upon new price generation, but at the end of a 10 usec window it computes the average price of the window. In §5.2, we quantify the overhead using our Click-based [29] implementation.

Parameter values: We set the average window size, d, to be twice the average RTT. A large window makes the system react to changes more slowly, but makes it more robust by reacting less aggressively to transient changes. The queue parameter α is set to 2, but to prevent the remaining link capacity (denominator of Eq. 2) from becoming negative,[4] we bound its minimum value to 1/2 the capacity. We verify the local stability of FCP using simulations in §5.1 under this setting.

5. EVALUATION

We answer four questions in this section:

1. How does FCP perform compared to other schemes? (§5.1)
2. What is the processing overhead of FCP? (§5.2)
3. Does FCP accommodate diversity at the end-points? (§5.3)
4. Does FCP accommodate diversity in the network? (§5.4)

[3]As a rough estimate, Intel Core i7 series have advertised performance of ∼100 GFLOPS. At 100 Gbps, this gives a budget of ∼500 FLOPS/packet.

[4]This can happen when the queue capacity is large compared to the current bandwidth delay product.

Case	new flows/sec	FCP utilization (%)	RCP utilization (%)
(a)	41.6	98.2	74.8
(b)	83.2	95.7	69.2
(c)	125	81.9	69.8

Table 1: Mixed flows: performance statistics

We use packet-level simulations using ns-2 and a Click [29]-based implementation of FCP for our evaluation.

5.1 Performance

First, we compare the performance of FCP with other schemes (RCP and XCP) using packet-level simulations. We then look at unique characteristics of FCP, including its fairness, preloading effectiveness, and stability using both simulations and our Click-based implementation. Finally, we look at FCP's performance under a wide range of scenarios. The results show that FCP provides fast convergence while providing more accurate feedback during convergence and is as efficient as other explicit congestion control algorithms in many cases.

5.1.1 Performance comparison

We demonstrate FCP's efficiency under various workloads.

Long Running Flows: We first compare the convergence dynamics of long-running flows. We generate flows with different round-trip propagational delays ranging from 25 ms to 625 ms. Each flow starts one second after another traversing the same 100 Mbps bottleneck in a dumbbell topology. Each flow belongs to a different sender, and all senders were assigned a budget of 1 $/sec. For RCP and XCP, we used the default parameter setting. Figure 3 shows the sending rate of each flow, queue size, and utilization for a) FCP, b) XCP, and c) RCP over time.[5] FCP's convergence is faster than RCP's and XCP's. XCP flows do not converge to the fair-share even at t=10 sec. In RCP, all flows get the same rate as soon as they start because the router gives the same rate to all flows. However, large fluctuation occurs during convergence because the total rate overshoots the capacity and packets accumulate in the queue when new flows start. Average bottleneck link utilization was 99% (FCP), 93% (XCP), and 97% (RCP).

Mixed flow sizes: We now study how FCP and RCP perform with mix flow sizes. All flows go through a common bottleneck of 100 Mbps. A long running flow starts at t=0.5 sec. FCP's long-running flow uses a unit budget. Short flows arrive as a Poisson process, and their size is Pareto distributed with a mean size of 30 KB and shape of 1.2. We vary the fraction of bandwidth that short flows occupy from 10% to 30%. Table 1 shows the average number of new short flows per second and the average link utilization from t=0.5 sec to t=30 sec for FCP and RCP. RCP's link utilization is much lower than that of FCP. This is because in RCP when short flows terminate, it takes an average RTT to update the new rate and an extra RTT for the long flow to ramp up, whereas FCP explicitly signals the termination of flow using negative preloading. FCP's utilization is high, but it becomes slightly lower as the number of short flows increase. This is because when flows terminate, even though they preload a negative value, it takes some time for the price to reflect due to the averaging window. In general, the utilization has a negative correlation with the amount of input budget variance per unit time. However, FCP handles mixed flow sizes much more gracefully and is much more efficient than RCP.

[5]The sending rate is averaged over a 100 ms. For XCP ($t \geq 3$), it is averaged over 500 ms since it was more bursty.

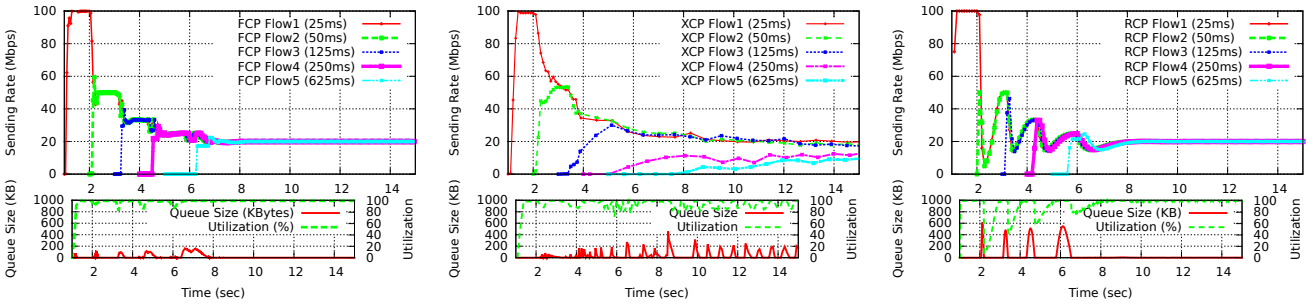


Figure 3: Convergence dynamics of a) FCP, b) XCP, and c) RCP: FCP achieves fast convergence and accurate rate control. When new flows arrive, XCP's convergence is slow, and RCP's rate overshoot is significant.

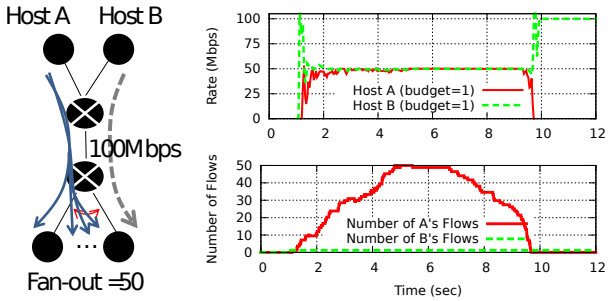


Figure 4: Fair-share is determined by sender's budget, not the number of flows.

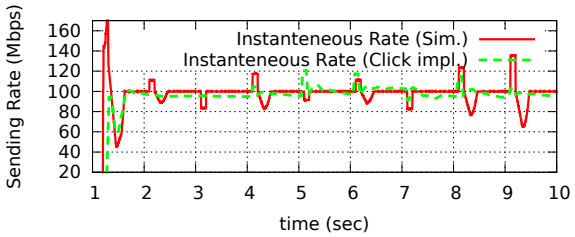


Figure 5: FCP is stable under random perturbation.

5.1.2 Fairness, stability, and preloading

To better understand FCP, we look at the behaviors specific to FCP. In particular, we look at the fairness, stability, and the preloading behavior. We show that 1) the fair-share is determined by the budget, 2) FCP is locally stable under random perturbations, and 3) preloading allows fast convergence and accurate feedback.

FCP's fairness is very different from traditional flow rate fairness. Two hosts with equal budget achieve the same throughput when the path price is the same regardless of the number of flows. To show this, we create two hosts A and B of equal budget, whose flows traverse the common bottleneck link (see Figure 4 left). Host A's flows of size 1 MB arrive randomly between t=[1,3], and Host B sends a long-running flow starting at t=1. Figure 4 (right) shows the total sending rate and the number of active flows. Host A's traffic, when present, gets 50% share regardless of the number of flows.

Local stability: A common method to prove the local stability is to show the stability of a linearized equation of the congestion controller near its equilibrium point [25]. However, the feedback equation at the equilibrium point of explicit congestion control algorithms is not continuous and the above method produces incorrect results [4].

Therefore, we demonstrate the stability by introducing random perturbations in our packet-level simulations and click-based implementation. We intentionally introduce errors in the price calculation and observe whether the system is able to return to correct and stable operation. We use a single link with a round trip delay of 100 ms with 100 Mbps of capacity. A long running flow starts at t=1.1 sec with a unit budget. At every second from t=2, we introduce random perturbation of [-30%, 30%] in the router's price calculation for 100 ms. For example, during the interval, t=[2,2.1] sec, the feedback price is consistently off from the correct price by a fixed fractional amount drawn randomly. Figure 5 shows the instantaneous sending rate of the flow in our simulation and click-based implementation. In both cases, FCP returns to equilibrium shortly after the perturbation.

How effective is preloading? So far, we have seen the performance of FCP with preloading. We now look at the benefit of preloading in isolation by comparing FCP with RCP that does not have preloading. We compare a FCP flow that continuously doubles its budget every 100 ms (one RTT) with RCP flows that double in flow count for a duration of 1 second. In both cases, the load (FCP's input budget and RCP's flow count) doubles every RTT.

Figure 6 shows the normalized incoming traffic at the bottleneck link while the load is ramping up from 1 to 1000 during the first second. We see that preloading allows fast convergence and accurate rate control; FCP's normalized rate is close to 1 at all times. With preloading end-hosts can rapidly increase or decrease a flow's budget without causing undesirable behaviors. On the other hand, RCP significantly overestimates the sending rate because RCP allocates bandwidth in very aggressively to mimic processor sharing.

Preloading with short flows: Preloading is also different from the step-change algorithm [27] for RCP. FCP's preloading allows end-points to explicitly account for short flows by requesting only the amount of resources it is going to consume on a packet-by-packet basis, which handles application's bursty traffic of known size (e.g, HTTP response in a web server) much more effectively. To highlight this, we generate two flows, a long flow and a short flow (25 KB), with a unit budget sharing a bottleneck with round-trip delay of 40 ms. Figure 7 (a) shows the sending rate for both flows at every 10 ms period when the short flow uses its entire budget, which mimics the behavior of the step-change algorithm. It preloads its entire budget, but does not send much traffic after the preload. As a result, the path price goes up and Flow 0 nearly halves its sending rate. This period of underutilization only ends after the duration of the averaging window (an average RTT in RCP) when the router adjusts the price to achieve full utilization. Figure 7 (b) shows the correct FCP behavior, which preloads the exact amount of budget it is going to use (compared to current budget consumption) and performs negative preloading at the end of the flow. In summary, FCP's preloading makes an end-point a much more active component in congestion control, ensuring high utilization and fast convergence.

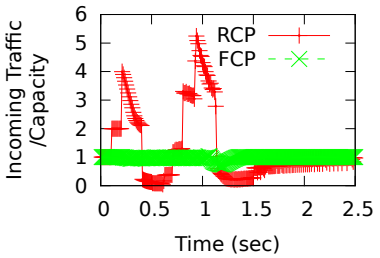
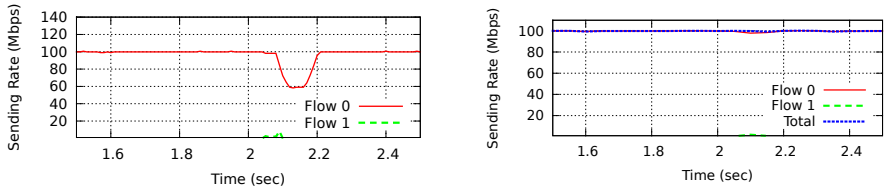


Figure 6: Preloading allows fast, graceful convergence.



(a) A misbehaving short flow

(b) Correct preloading

Figure 7: Preloading achieves high link utilization.

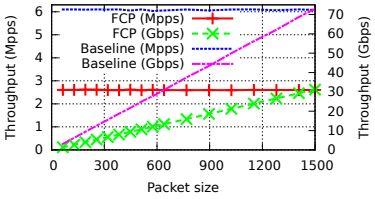


Figure 8: Forwarding throughput

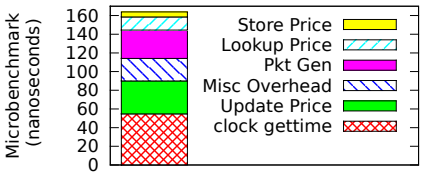


Figure 9: Microbenchmark

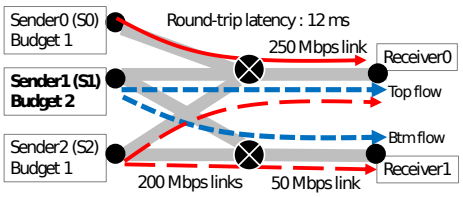


Figure 10: Topology

5.2 Overhead

Using our Click implementation, we measure the packet forwarding throughput and perform micro-benchmarking with a single-thread Click running on an Intel Xeon L5640 2.26 GHz core. Our implementation holds past price history up to 500 ms and updates the history every 10 usec, as described in §4. The total memory use was about 72MB with the heap size being 54MB. To measure the computational overhead of FCP's algorithm, we intentionally eliminated the I/O. Thus, our experiment only generates packets inside Click, runs through the algorithm, and discards the packets.

Figure 8 shows the throughput (in Mpackets per second and in Gbits per second) of our implementation and of baseline Click without FCP by varying packet size (from 64 bytes to 1500 bytes). The baseline Click gives a throughput of 6.1 Mpps (3.1 Gbps @ 64 bytes), and our FCP implementation performs at 2.6 Mpps (1.25 Gbps @ 64 bytes) using a single core. Upon inspection of the micro-benchmark (Figure 9), we observe that only a fraction of cycles (33%) are spent performing the FCP-related computation. The rest of the cycles are spent reading the system time when the packet was received (33%), generating the packets (18.5%), and other Click overhead (14.8%), such as recycling the internal packet pool and driving the packet through the list of Click elements. Such overhead can be easily removed in hardware implementations. For example, packet reception timestamp can be obtained via the NIC. Ignoring such overhead, FCP's forwarding throughput can scale to 7.8 Mpps (3.78 Gbps @ 64 bytes).

Although the overhead is not prohibitive, one might imagine this would hinder FCP's deployment on high-speed links. We believe that it is possible to use virtual links/queues and receiver-side-scaling (RSS) to scale the performance significantly. However, we leave this as future work, focusing on the core algorithm in this paper.

5.3 End-point Flexibility

We now demonstrate the flexibility of FCP's end-point outlined in §3.2 using packet-level simulations. For comparison, we use the common topology of Figure 10 unless otherwise noted. Sender 0 (S0) transmits a flow to receiver 0. Both S1 and S2 have a flow to receiver 0 and receiver 1. S0 and S2 have a budget of 1 $\$/sec$, and S1 has twice as much.

Equal-budget (baseline) splits the budget equally among flows within a host. For example, S1 splits its budget in half; the top flow (to receiver 0) and the bottom flow (to receiver 1) get 1 $\$/sec$ each. Figure 11 shows the instantaneous sending rate of each sender with S1's rate broken down. It shows FCP achieves weighted bandwidth allocation. The top 250 Mbps bottleneck link's total input budget is 2.5 $\$/sec$. Because S0 and S1's top flow use 1 $\$/sec$, their throughput is 100 Mbps.

Equal throughput: Now, S1 changes its budget assignment to equal-throughput where it tries to achieve equal throughput on its flows, while others still use equal-budget assignment. S1 starts with the equal-budget assignment, and reassigns budget every two average RTTs, increasing the budget of a flow whose rate is less than the average rate. Figure 12 shows the result. At steady state, S1's two flows achieve the same throughput of 38.7 Mbps. A budget of 0.27 $\$/sec$ is assigned to the top flow, and 1.73 $\$/sec$ to the bottom.

Max-throughput: S1 now tries to maximize its total throughput, while others still use the equal-budget assignment. We implement this using gradient ascent. S1's flows start with equal budget, but at every two average RTT, it performs an experiment to change the budget assignment. It chooses a flow in round robin fashion and increases its budget by 10% while decreasing others uniformly to preserve the total budget allocation. After two average RTTs, it compares the current throughput averaged over an RTT with the previous result, and moves towards the gradient direction. The algorithm stops when the throughput difference is less than 0.5%, but restarts when it observes a change in the path price.

Figure 13 shows the result. S1's total throughput converges at 150.6 Mbps, and the assigned budget for the top flow (X) converges at 1.56 $\$/sec$. Figure 14 a) shows the theoretical throughput versus the budget assignment, X. The theoretical maximum throughput is 151.6 Mbps at $X = 1.68$ When more (less) budget is spent on X than this, the top (bottom) bottleneck link's price goes up (down), and the marginal utility becomes negative. Figure 14 b) shows such non-linear utility (rate per unit budget) curve for the two flows.

Background flows: FCP also supports background flows, which by definition is a flow that only occupies bandwidth if there's no other flow competing for bandwidth. This can be achieved with a flow having a very small assigned budget compared to other flows. For this, each host uses 1/10000 of the normal flow's budget towards

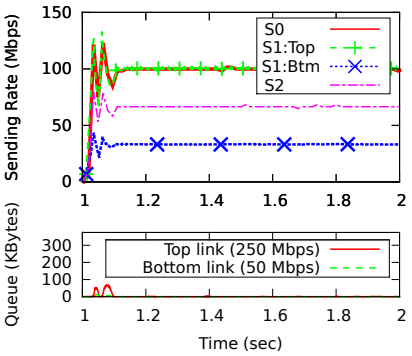


Figure 11: Equal-budget flows

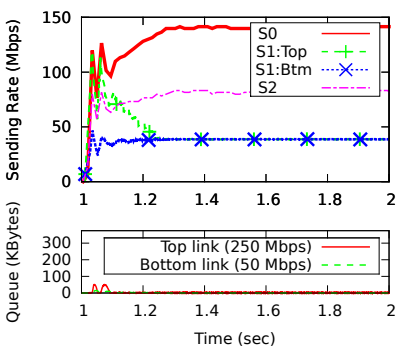


Figure 12: Equal-throughput flows

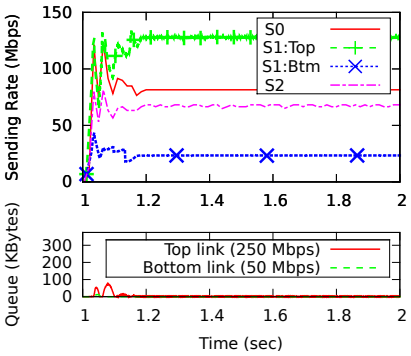


Figure 13: Max-throughput flows

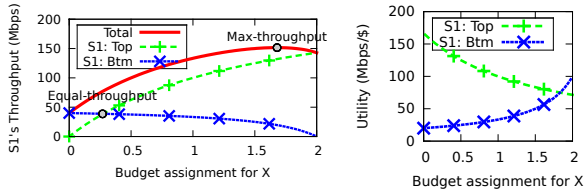


Figure 14: Theoretical (a) throughput and utility (b) versus budget assignment X for flow S1's top flow.

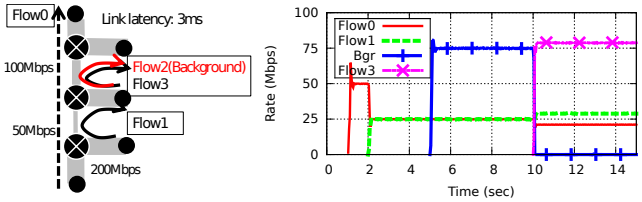


Figure 15: Background flows only take up the spare capacity. The sending rate is averaged over 40 ms.

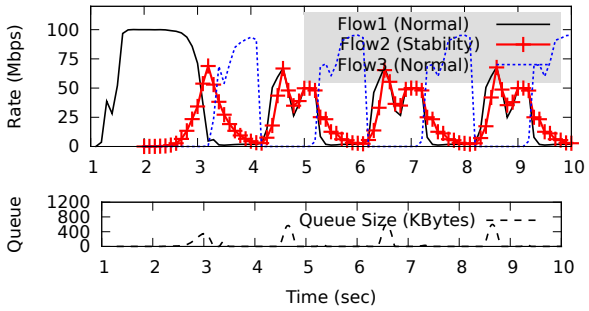


Figure 16: Support for bandwidth stability

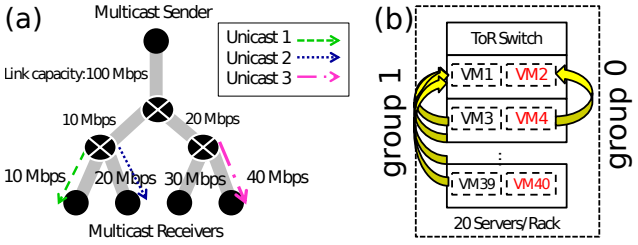


Figure 17: Topologies and flow patterns of experiments

background flows. Using the topology in Figure 15, we run a background flow (Flow 2) with three normal flows (Flow 0, 1, and 3) with a unit budget. Flow 0 starts at t=1 and immediately occupies the 50 Mbps bottleneck link (Figure 15). Flow 1 arrives at t=2 and shares the bottleneck with Flow 0. At t=5, the background flow (Flow 2) starts and immediately occupies the remaining 75 Mbps of the 100 Mbps link. Note this did not affect Flow 0's rate. Flow 3 arrives at t=10 with unit budget and drives out the background flow. Note that now the 100 Mbps link also became a bottleneck, and Flow 0 is getting a smaller throughput than Flow 1. This is because Flow 0 now pays the price of the two bottleneck links combined.

5.4 Network Flexibility

FCP's local control and aggregation combined with preloading allow FCP to implement a variety of QoS schemes and support a different network service model by modifying the router's behavior, while being compatible with the original FCP. To highlight this, we demonstrate the features described in §3.2.

Bandwidth stability: We first demonstrate the bandwidth stability feature. Figure 16 shows the sending rate of a stability flow (`Flow 2`) compared to a normal flow (`Flow 1`) under changing network conditions. Flow 2 starts at t=2 and slowly ramps up doubling its assigned budget at every RTT (100 ms). Cross traffic (Flow 3, dashed line) that has 50 times the budget of Flow 1 repeats a periodic on/off pattern starting from t=3. Flow 1's sending rate (solid black line) changes abruptly when the cross traffic joins and leaves, but the stability flow reacts slowly because its price does not change by more than twice in any direction during the averaging window.

Multicast congestion control: FCP's flexibility even enables the network to support congestion control with a different service model, such as multicast (§3.2). We use the topology of Figure 17 (a) and generate a multicast flow from the top node to the bottom four receivers. The link capacity varies by link from 10 to 100 Mbps. We also introduce three competing unicast flows (Unicast 1, 2, and 3) to vary the load over time, and show that the multicast flow dynamically adapts its rate.

Figure 18 shows the result. A multicast flow starts at t=1 and saturates the bottleneck link capacity of 10 Mbps. Three unicast flows then start sequentially at t=2, 2.5, 3. When `Unicast 1` arrives, it equally shares the bottleneck bandwidth. As other unicast flows arrive, other links also become a bottleneck and their price goes up. As a result, the multicast flow's price (the sum of all link prices) goes up and its sending rate goes down. At steady state, the multicast flow's throughput is around 4 Mbps. The unicast flows take up the remaining capacity (e.g., unicast 3's rate is 36 Mbps).

Deadline support: FCP can offer D^3-style [49] deadline support using differential pricing. Deadline flows are guaranteed a fixed price when admitted. We use the minimum price (described in §4)

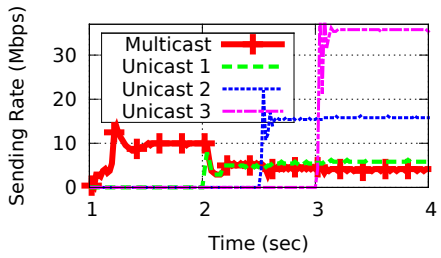


Figure 18: Multicast flow adapts to congestion.

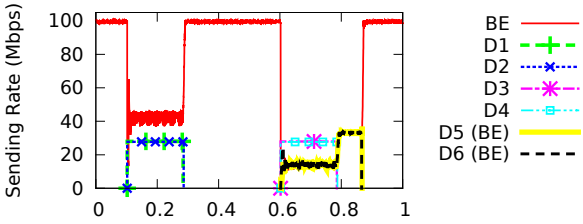


Figure 19: Deadline flows

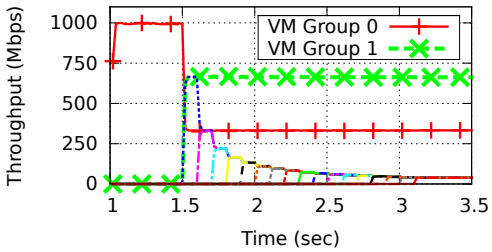


Figure 20: Aggregate control

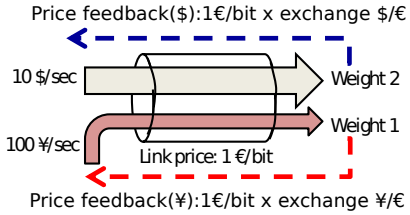


Figure 21: Dynamic value translation.

as the fixed price. At the beginning, a deadline flow preloads the amount required to meet the desired rate at once. If the routers on the path can accommodate the new deadline flow, they return this minimum price as feedback. The deadline flow is then able to send at the desired rate to meet the deadline. Otherwise, routers give the normal pricing and treat the flow as a best effort flow. Our results in Figure 19 show both cases.

Figure 19 shows the instantaneous rates of deadline and best effort flows going through a 100 Mbps bottleneck link with a round-trip delay of 2 ms. The queue is set to admit up to 80 Mbps of deadline flows and assigns at least 20 Mbps to best effort flows in a work conserving manner. A best effort (BE) flow starts at the beginning and saturates the link. At t=0.1 sec, two deadline flows (D1 and D2) requiring 30 Mbps of throughput arrive to meet a flow completion time of ∼200 ms. Because the link can accommodate the deadline flows, they both get admitted and complete within the deadline. At t=0.6 sec, four deadline flows (D3 to D6) arrive with the same requirement. However, the network can only accommodate two deadline flows (D4 and D5). The other two (D5 and D6) receive the best effort pricing and additionally preload to achieve their fair-share in the best effort queue.

6. BUDGET MANAGEMENT

So far, we assumed the budget is allocated in a centralized manner. However, FCP also allows decentralized budget management. In particular, FCP allows networks to dynamically translate the value of a budget belonging to different flow groups or assigned by different domains. The former allows aggregate congestion control between groups of flows, and the latter enables distributed budget assignment between mutually untrusted domains.

Aggregate control: FCP allows the network to perform aggregate congestion control over an arbitrary set of flows so that each group in aggregate attains bandwidth proportional to its weight. This, for example, can be used to dynamically allocate bandwidth between multiple tenants in a data-center [44].

To achieve this, we view each group as having its own currency unit whose value is proportional to the weight of the group and is inversely proportional to the aggregate input budget of the group. When giving feedback, a router translates its price into the group's current currency value (Figure 21). For example, if flow group A has a weight of 2 and B has a weight of 1, and their current input budgets are, respectively, 10 $\$/sec$ and 100 $¥/sec$, A's currency has more value. To estimate the input budget for each group of flows, we use the `balance` field. Each router keeps a separate input budget for each group. It also keeps the aggregate input budget using its own link's price history and updates the price as in the original scheme using Equation 2. Thus, we calculate the normalized exchange rate E_G for flow group G as:

$$E_G(t) = \frac{w_G}{\sum_{H \in Group} w_H} / \frac{I_G(t)}{\sum_{H \in Group} I_H(t)}$$

And adjust the price feedback for packet s as:

$$price(s) = price(s) + p(t) \cdot E_G(t)$$

where $p(\cdot)$ is defined in Equation 2.

We apply this to allocate resources between tenants in a data-center. We perform a packet-level simulation, using the flow pattern of Figure 17 (b), similar to the example in [44]. Two tenants share 20 servers in a rack, each having two virtual machines (VMs). VM group 0 belongs to tenant 0 and group 1 to tenant 1. All links run the algorithm above, and the cloud provider assigns a weight of two to tenant 1 and a unit weight to tenant 0. Each tenant can allocate a budget to each of its own VMs independently. Here, we assume they assign a unit budget to all VMs. Figure 20 shows the result with the aggregate throughput of each group and individual flows. From t=1.5 sec, VM group 1's 20 flows start in sequence. The result highlights two main benefits: 1) Regardless of the number of flows in the group, the aggregate flows get bandwidth proportional to their weights; group 1 gets twice as much as group 0. 2) The weight of flows within each group is preserved; group 1's individual flows (dotted lines), which have equal budget, achieve equal throughput.

Inter-domain budget management: In an Internet scale deployment, budget assignment must be made in a distributed fashion. One solution is to let each ISP or domain assign budget to its customers without any coordination, and rely on dynamic translation of the value. Here, we outline the approach.

When a packet enters another domain, the value the packet is holding (balance field in the header) can be translated to the ingress network's price unit. One can use a relatively fixed exchange rate that changes slowly or a dynamic exchange rate similar to the aggregate congestion control above. This allows the provider (peering) network to assign bandwidth to its customers (peers) proportional to their weight. For the price feedback to be represented in the unit of the sender's price, the data packet must carry an additional field

for the exchange rate. The scheme protects the provider's network from malicious customer networks that intentionally assign large amount of budget in an attempt to obtain more resources, because their budget's actual value will be discounted.

7. CONCLUSION

This paper explores an important open problem of accommodating diverse application requirements in congestion control. While there have been a number of studies to accommodate diverse styles of communication in Internet architectures [20, 24], little research shows how to accommodate diverse behaviors in congestion control. This problem is further exacerbated by recent advances in router-assisted congestion control, which makes congestion control inherently much less flexible. In this paper, we address this challenge by introducing a flexible framework for congestion control, called FCP, that allows various strategies and algorithms to coexist within the same network. FCP gives more control at the end-point and allows the network to perform resource allocation with the end-point's input. We demonstrate that this not only enables end-points to optimize their own objective, but also allows various quality-of-service features and resource management policies to be introduced in the network.

Acknowledgments

We thank Aaron Gember, Saul St. John, Hyeontaek Lim, David Naylor, Jeff Pang, and Vyas Sekar for their valuable comments and suggestions on earlier versions of this paper. We also thank our shepherd Arvind Krishnamurthy and anonymous reviewers for their feedback. This research was supported in part by the National Science Foundation under awards CNS-1040801, CNS-1040757, CNS-0905134, and CNS-0746531.

References

[1] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet impasse through virtualization. *IEEE Computer*, 38, Apr. 2005.

[2] S. Athuraliya, S. Low, V. Li, and Q. Yin. Rem: active queue management. *IEEE Netw.*, 15(3):48 –53, May 2001.

[3] A. Balachandran et al. A quest for an internet video quality-of-experience metric. In *Proc. ACM HotNets*, 2012.

[4] H. Balakrishnan, N. Dukkipati, N. McKeown, and C. Tomlin. Stability analysis of explicit congestion control protocols. *IEEE Commun. Lett.*, 11(10), 2007.

[5] H. Balakrishnan, H. S. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proc. ACM SIGCOMM*, Sept. 1999.

[6] D. Bansal and H. Balakrishnan. Binomial Congestion Control Algorithms. In *IEEE Infocom*, Anchorage, AK, Apr. 2001.

[7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Service. RFC 2475 (Informational), Dec. 1998.

[8] C. Borchert, D. Lohmann, and O. Spinczyk. CiAO/IP: a highly configurable aspect-oriented IP stack. In *Proc. ACM MobiSys*, June 2012.

[9] P. G. Bridges, G. T. Wong, M. Hiltunen, R. D. Schlichting, and M. J. Barrick. A configurable and extensible transport protocol. *IEEE ToN*, 15(6), Dec. 2007.

[10] B. Briscoe. Flow rate fairness: dismantling a religion. *ACM SIGCOMM CCR*, 37(2), Mar. 2007.

[11] B. Briscoe, A. Jacquet, C. Di Cairano-Gilfedder, A. Salvatori, A. Soppera, and M. Koyabe. Policing congestion response in an internetwork using re-feedback. In *Proc. ACM SIGCOMM*, 2005.

[12] C. Courcoubetis, V. A. Siris, and G. D. Stamoulis. Integration of pricing and flow control for available bit rate services in ATM networks. In *Proc. IEEE GLOBECOM*, 1996.

[13] J. Crowcroft and P. Oechslin. Differentiated end-to-end internet services using a weighted proportional fair sharing TCP. *ACM SIGCOMM CCR*, 28, 1998.

[14] D. Damjanovic and M. Welzl. MulTFRC: providing weighted fairness for multimedia applications (and others too!). *ACM SIGCOMM CCR*, 39, June 2009.

[15] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. In *Proc. ACM SIGCOMM*, 2011.

[16] N. Dukkipati, M. Kobayashi, R. Zhang-shen, and N. Mckeown. Processor sharing flows in the Internet. In *Proc. IWQoS*, 2005.

[17] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. ACM SIGCOMM*, 2000.

[18] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1, Aug. 1993.

[19] B. Ford. Structured streams: a new transport abstraction. In *Proc. ACM SIGCOMM*, 2007.

[20] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox. Intelligent design enables architectural evolution. In *Proc. ACM HotNets*, 2011.

[21] R. J. Gibbens and F. P. Kelly. Resource pricing and the evolution of congestion control. *Automatica*, pages 1969–1985, 1999.

[22] S. Ha, I. Rhee, and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.*, 42, July 2008.

[23] D. Han, A. Anand, A. Akella, and S. Seshan. RPT: Re-architecting loss protection for content-aware networks. In *Proc. 9th USENIX NSDI*, Apr. 2012.

[24] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste. XIA: Efficient support for evolvable internetworking. In *Proc. USENIX NSDI*, 2012.

[25] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *Proc. ACM SIGCOMM*, Pittsburgh, PA, Aug. 2002.

[26] F. Kelly. Charging and rate control for elastic traffic. *Eur. Trans. Telecommun.*, 1997.

[27] F. Kelly, G. Raina, and T. Voice. Stability and fairness of explicit congestion control with small buffers. *ACM SIGCOMM CCR*, 2008.

[28] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49(3), 1998.

[29] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, Aug. 2000.

[30] S. Kunniyur and R. Srikant. End-to-end congestion control schemes: utility functions, random losses and ecn marks. *IEEE/ACM ToN*, 11(5), 2003.

[31] A. Lakshmikantha, R. Srikant, N. Dukkipati, N. McKeown, and C. Beck. Buffer sizing results for rcp congestion control under connection arrivals and departures. *SIGCOMM Comput. Commun. Rev.*, 39(1), Dec. 2009.

[32] A. Li. RTP Payload Format for Generic Forward Error Correction. RFC 5109 (Proposed Standard), Dec. 2007.

[33] S. H. Low and D. E. Lapsley. Optimization flow control. i. basic algorithm and convergence. *IEEE/ACM Trans. Netw.*, 7(6):861–874, Dec. 1999.

[34] R. T. Ma, D. M. Chiu, J. C. Lui, V. Misra, and D. Rubenstein. Price differentiation in the kelly mechanism. *SIGMETRICS PER*, 40(2), Oct. 2012.

[35] J. K. MacKie-Mason and H. R. Varian. Pricing the internet. Computational Economics 9401002, EconWPA, Jan. 1994.

[36] L. Massoulié and J. Roberts. Bandwidth sharing: objectives and algorithms. *IEEE/ACM Trans. Netw.*, 10(3):320–328, June 2002.

[37] P. Natarajan, J. R. Iyengar, P. D. Amer, and R. Stewart. SCTP: an innovative transport layer protocol for the web. In *Proc. World Wide Web*, 2006.

[38] A. Odlyzko. Paris metro pricing: The minimalist differentiated services solution. In *Proc. IWQoS*, 1999.

[39] P. Patel, A. Whitaker, D. Wetherall, J. Lepreau, and T. Stack. Upgrading transport protocols using untrusted mobile code. In *Proc. ACM SOSP*, 2003.

[40] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica. Faircloud: sharing the network in cloud computing. In *Proc. ACM SIGCOMM*, 2012.

[41] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren. Cloud control with distributed rate limiting. In *Proc. ACM SIGCOMM*, 2007.

[42] S. Shenker, D. Clark, D. Estrin, and S. Herzog. Pricing in computer networks: reshaping the research agenda. *ACM SIGCOMM CCR*, 26(2), Apr. 1996.

[43] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the testbed? In *Proc. 9th USENIX OSDI*, Vancouver, Canada, Oct. 2010.

[44] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *Proc. 8th USENIX NSDI*, 2011.

[45] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: achieving approximately fair bandwidth allocations in high speed networks. In *Proc. ACM SIGCOMM*, 1998.

[46] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz. OverQoS: An overlay based architecture for enhancing Internet QoS. In *Proc. 1st USENIX NSDI*, San Francisco, CA, Mar. 2004.

[47] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter TCP (D2TCP). In *Proc. ACM SIGCOMM*, 2012.

[48] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: a mechanism for background transfers. *SIGOPS Oper. Syst. Rev.*, 36, Dec. 2002.

[49] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: meeting deadlines in datacenter networks. In *Proc. ACM SIGCOMM*, 2011.

[50] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *Proc. 8th USENIX NSDI*, Boston, MA, Apr. 2011.

[51] Y. Yang and S. Lam. General aimd congestion control. In *Proc. ICNP*, 2000.

[52] Y. Yi and M. Chiang. Stochastic network utility maximisation - a tribute to kelly's paper published in this journal a decade ago. *European Transactions on Telecommunications*, 19(4):421–442, 2008.