# High Speed Networks Need Proactive Congestion Control

Lavanya Jose⋆, Lisa Yan⋆, Mohammad Alizadeh†, George Varghese‡, Nick McKeown⋆, Sachin Katti⋆
⋆Stanford University, †Massachusetts Institute of Technology, ‡Microsoft Research

{lavanyaj,yanlisa,nickm,skatti}@stanford.edu
alizadeh@mit.edu, varghese@microsoft.com

## ABSTRACT

As datacenter speeds scale to 100 Gb/s and beyond, traditional congestion control algorithms like TCP and RCP converge slowly to steady sending rates, which leads to poorer and less predictable user performance. These *reactive* algorithms use congestion signals to perform gradient descent to approach ideal sending rates, causing poor convergence times. In this paper, we propose a *proactive* congestion control algorithm called PERC, which explicitly computes rates independently of congestion signals in a decentralized fashion. Inspired by message-passing algorithms with traction in other fields (e.g., modern Low Density Parity Check decoding algorithms), PERC improves convergence times by a factor of 7 compared to reactive explicit rate control protocols such as RCP. This fast convergence reduces tail flow completion time (FCT) significantly in high speed networks; for example, simulations of a realistic workloads in a 100 Gb/s network show that PERC achieves up to 4× lower $99^{th}$ percentile FCT compared to RCP.

**Categories and Subject Descriptors**
**C.2.1 [Computer-Communication Networks]:** Network Architecture and Design

**Keywords**
Datacenter network, Flow scheduling, Congestion control

## 1. INTRODUCTION

Over the past decade, link speeds have grown steadily from 1 Gb/s to 10 Gb/s and now 100 Gb/s. Network transfers are completing faster and in fewer RTTs; given this trend, allocating flows their proper rates as quickly as possible becomes a priority. More precisely, we assert that *convergence time* must become a primary, low-level metric for congestion control in high speed networks. Convergence time impacts higher level flow metrics like flow completion time (FCT): at higher link rates, a majority of flows last only a few RTTs, and if they take longer to converge to ideal rates, some of the flows end up spending more time in the network than needed, affecting the tail FCTs.

Convergence time has been a secondary consideration in the design of today's congestion control algorithms. Existing protocols like TCP, DCTCP [1], RCP [12], XCP [18], etc. are *reactive* in nature, iteratively reacting to congestion signals from the network. For example, TCP treats packet drops as a signal of network congestion and uses AIMD to iteratively converge to a sending rate [16]. Reactive algorithms fundamentally have long convergence times because they need to measure congestion signals before they can react to them; moreover, rate adjustment is generally performed over several iterations, resembling a gradient descent procedure which gradually approaches the target sending rates.[1]

In the early 2000s, researchers recognized that congestion control algorithms such as TCP which uses binary congestion signals (e.g., packet drops), converge slowly. This spurred the development of algorithms designed to converge faster than TCP, such as XCP [18] and RCP [12]; in these algorithms, the routers along a flow's path calculate a current rate estimate and explicitly feed it back to the source.

Despite these improvements, XCP and RCP are reactive gradient descent algorithms too; both advertise rate changes (in XCP's case, congestion window changes) every RTT after measuring how the previous iteration has performed using traffic and/or queue measurements. Because of the interdependency between flows on different paths sharing congested links, the routers do not know the correct final converged rate; they only update the rate feedback to *react* to the most recent information they have passively gained about local congestion. Furthermore, reactive algorithms have to adjust rates cautiously to keep the system stable. XCP, RCP and other reactive algorithms use gradient descent steps towards the final rates, carefully chosen to strike a balance between convergence speed and stability.

This paper advocates *proactive* congestion control mechanisms that *explicitly calculate sending rates based on flow*

[1]Refer to [24, 9, 19] for elaboration on the connection between rate control algorithms and distributed optimization.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*HotNets '15* November 16–17 2015, Philadelphia, PA USA
Copyright 2015 ACM 978-1-4503-4047-2 ...$15.00.

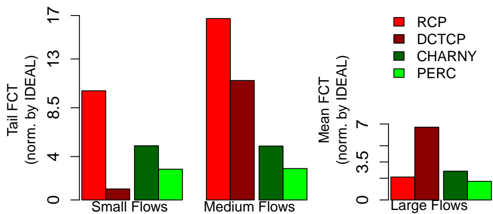



Figure 1: **FCT on 100 Gb/s link with 60% load and 120 $\mu$s RTT: $99^{\text{th}}$ percentile for small (< 10 KB) and medium (10 − 1000 KB) flows, and mean for large (> 1 MB) flows. Normalized by respective statistics for ideal instantaneous max-min rate allocation.**

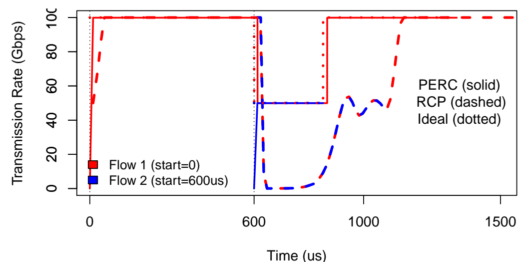



Figure 2: **Evolution of datarates over time in RCP and PERC. The 100 Gb/s link initially has Flow 1 with length 10K packets. At 600 $\mu s$, short Flow 2 begins transmitting 1K packets. The red and blue, step like, solid curves are the rates with PERC.**

*demands*. Unlike reactive algorithms adjust rates gradually based on indirect signals like *local* congestion at each link, proactive algorithms compute optimal rates based on *global* information about which flows are active at each link and their rate demands. While proactive algorithms also need multiple iterations to compute the optimal rates in a decentralized fashion, the use of global information allows significantly fewer iterations; each iteration can also be much faster because proactive schemes can skip the slow measurement steps of reactive congestion-based algorithms.

In high speed networks, a majority of the flows are short enough to complete in a few to a few tens of RTTs. We show that in such scenarios, proactive algorithms achieve significantly lower tail FCTs than reactive algorithms because of their quick convergence.

As a quick preview of our results, consider the following simple experiment where we run a standard web-search workload [1] on a single 100 Gb/s link with a 120 $\mu s$ round-trip delay. Figure 1 shows the tail FCT for small and medium sized flows, and the mean FCT for large flows. The results are normalized with respect to an ideal scheme that instantaneously allocates the optimal max-min fair rates. For now we focus on PERC, our proactive design, and RCP, a representative reactive algorithm. We discuss other schemes and different network settings later (Section 4).

We see that for small and medium flows, RCP's tail FCT is almost four times that of PERC. Thus PERC provides more *predictable delays* for these flows than the reactive RCP. Intuitively, the large tail FCT with RCP is caused by some flows getting smaller rates when new flows arrive due to poor convergence times. For large flows these transient periods of under-utilization have less impact, but they adversely impact small and medium sized flows. This intuition is confirmed by a second experiment shown in Figure 2. Notice the sharp dip in the sending rate for RCP at 600 $\mu s$ as the second flow starts and causes congestion, and the slow climb to max-min fair rates. During the climb, RCP underutilizes the link.

We proceed by presenting a model for congestion control in Section 2, describing PERC in Section 3, and providing a detailed evaluation in Section 4 .

## 2. MODELS

All congestion algorithms, whether reactive or proactive, have a target set of flow rates to which they are trying to converge. As a concrete example, we focus on congestion control algorithms that target *max-min fairness* [5]. Other generalizations such as co-flow sharing [10] are possible, and the faster convergence times of proactive algorithms can also benefit these algorithms.

Consider a network with a static set of flows and links, where each flow $a$ traverses a path (i.e., set of links), $P(a)$, and each link $l$ has a set of flows, $F(l)$, that pass through it. If each flow $a$ sends at a rate $r_a$, then we define a *feasible* rate allocation as one that satisfies the link capacity constraints: $\sum_{a \in F(l)} r_a \leq C_l$, for all links $l$. A *max-min fair allocation* is a feasible allocation in which for any flow $a$, increasing its rate $r_a$ forces a decrease in the rate for a different flow $a'$, where $r_a \geq r_{a'}$. A defining characteristic for such an allocation is that for every flow $a$, there is at least one *bottleneck link* $l$ such that:

- Bottleneck link $l$ is saturated: $\sum_{a' \in F(l)} r_{a'} = C_l$.
- Flow $a$ sends at the *fair-share rate* $\lambda_l$ of link $l$, defined as the maximum rate among all flows in $F(l)$: $r_a = \lambda_l = \max_{a' \in F(l)} r_{a'}$.

It can be shown that for a given set of links and flows, there exists a unique max-min fair allocation [5]. In a dynamic network, the max-min allocation changes as new flows enter the network and old flows exit. We say that a max-min fair algorithm has *converged* when all flows are sending at the fair-share rate of their bottleneck links.

### 2.1 Reactive congestion control: RCP

We use RCP as a canonical reactive congestion control algorithm [12] where each link $l$ explicitly computes a rate $R_l$ and advertises this rate to each flow $a \in F(l)$. Each flow $a$ sends at the rate $r_a = \min_{l \in P(a)} R_l$, and eventually each rate $R_l$ converges to the fair-share rate $\lambda_l$. While other reactive rate control algorithms for max-min fairness such as XCP [18] have been proposed, we focus on RCP which has superior FCTs across all flow sizes [12, 20].

RCP is reactive because links adjust their advertised rates $R_l$ based on congestion signals, such as the input traffic rate $y_l(t)$ and queue length $q_l(t)$ at the link at time $t$. Every RTT, $d$, RCP updates $R_l$ at each link as follows:

$$R_l(t) = R_l(t-d)\left(1 + \frac{\alpha(C_l - y_l(t)) - \beta\frac{q_l(t)}{d}}{C_l}\right),$$

where $C_l$ is the link capacity, $y_l(t)$ and $q_l(t)$ are measured over the last update interval, $d$, and $\alpha$ and $\beta$ are positive parameters. The intuition behind this equation is that if there is spare capacity (i.e., $C_l - y_l(t) > 0$), $R_l$ is increased to distributed the remaining capacity among the current flows. Conversely, if the link is congested or there is queue buildup, $R_l$ is reduced. RCP is max-min fair; that is, all $R_l$ advertisements converge to the fair-share rates, $\lambda_l$, and thus flow rates converge to the max-min fair allocation.

RCP's update equation highlights the two phases of reactive congestion control algorithms:

In the *measurement phase*, the algorithm receives noisy congestion signals and must average out the noise over several packets. RCP measures queue size $q_l(t)$ and input traffic arrival rate $y_l(t)$ over an RTT $d$.

In the *reaction phase*, the algorithm takes a "step" to tune the rates in reaction to signals from the measurement phase, similar to a gradient descent algorithm moving in the direction of the gradient towards the optimal point. The time that the algorithm takes to converge depends on the number and size of such tuning steps. In RCP, the step sizes of the update equation are controlled by the $\alpha$ and $\beta$ parameters. In most reactive algorithms there is a tradeoff between convergence speed and stability; for example, RCP can try to converge quickly by taking large steps towards the target rate, but this risks perpetual overshooting and system instability.

## 2.2 Proactive congestion control

We focus on a proactive congestion control algorithm that calculates the max-min fair rates for the flows directly. This differs from reactive congestion in two ways: first, there is no measurement phase; the rates are calculated independently of congestion signals like queues and traffic volume. Thus, rate calculation is only limited by the time it takes for the network to register a change in the set of active flows — such as a flow arrival or departure — which is itself proportional to the propagation delay. Second, there is no notion of a step size as in gradient descent; rates are calculated explicitly based on which flows are active, thus avoiding the gradual adjustments that a reactive algorithm needs to converge to the target rates. The lack of a measurement phase and fast, explicit rate calculations help proactive algorithms converge very quickly.

An obvious way to implement proactive congestion control is via a centralized rate allocator, which knows all of the flows' sending rates and their paths through the network. When a flow joins or leaves the system, the allocator adjusts the rates of all affected flows. However, such a centralized scheme may be difficult to scale; thus, for the rest of this paper, we pursue a *distributed* proactive congestion control scheme based on *message passing* [21]. Each flow $a$ exchanges rate information with each link in its path $P(a)$, and asynchronously each link $l$ returns its fair share rate based on its current knowledge of the flow set $F(l)$. We now explain our algorithm (PERC) in detail.

# 3. PROACTIVE EXPLICIT RATE CONTROL

We first describe a synchronous version of PERC and its convergence behavior. We then discuss a practical asynchronous implementation, which we evaluate in Section 4.

## 3.1 Synchronous PERC

As in RCP, a link $l$ in PERC computes a fair share rate, $f_l$, that it then advertises to all flows $F(l)$ traversing it. Each flow $a$ sends at the rate $r_a = \min_{l \in P(a)}\{f_l\}$; in addition, it also advertises a rate *demand*, $d_l^a$, for each link $l \in P(a)$ within its packet headers. PERC is a proactive congestion control algorithm: it does not measure congestion signals, nor adjust its advertised rates in incremental steps. Instead, the fair share rate $f_l$ at each link $l$ is computed explicitly by solving an equation involving the link capacity and the demand information conveyed by the flows in packet headers.

The algorithm is inspired by message passing schemes [21] that are popular for various inference tasks such as for decoding Low Density Parity Check Codes [23]. Each flow $a$ calculates its demand $d_l^a$ per link as the sending rate it would have used had link $l$ been absent from its path, or simply:

$$d_l^a = \min_{l' \in P(a), l' \neq l} \{f_{l'}\}. \quad (1)$$

Single-hop flows demand an infinite rate. The demand is essentially the largest rate at which the flow can send,[2] ignoring any constraint from link $l$.

Meanwhile, each link $l$ picks its advertised fair share rate $f_l$ to be the max-min fair share rate for the link, given the advertised flow demands. More precisely, it computes $f_l$ as the solution to the following equation:

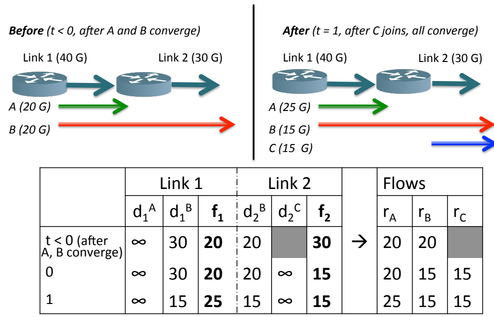
$$\sum_{a \in F(l)} \min(d_l^a, f_l) = C_l. \quad (2)$$

Notice that if flow $a$ has demand $d_l^a \leq f_l$, then it is bottlenecked elsewhere and can send at its demanded rate $d_l^a$. On the other hand, if flow $a$'s demand is higher than $f_l$, then link $l$ is responsible for assigning it a fair share of the link. It is not difficult to see that $f_l$ calculated above maximizes the fair share rate for flows bottlenecked at link $l$.

In the event that the aggregate flow demand is less than the link capacity (i.e., $\sum_{a \in F(l)} d_l^a < C_l$), Equation 2 does not have a solution. In this case, the link is not bottlenecked and must advertise that it has spare capacity to all flows in $F(l)$. To do so, a non-bottlenecked link $l$ simply advertises $f_l = C_l$ to signal to all flows in $F(l)$ that there is room to increase their sending rates. When the aggregate flow demand equals the link capacity, the link still advertises $C_l$.

In the synchronous version of PERC, flows and link alternate expressing demands and advertising fair share rates, respectively. We define an *iteration* of synchronous PERC as one single exchange of flow and link information. In each iteration, the flows compute their demands from the fair

[2] For simplicity, we assume here that all flows are bottlenecked in the network. If a flow is limited at the host or application, it can simply demand a lower rate.




| | Link 1 | | | Link 2 | | | | Flows | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $d_1^A$ | $d_1^B$ | $f_1$ | $d_2^B$ | $d_2^C$ | $f_2$ | | $r_A$ | $r_B$ | $r_C$ |
| t < 0 (after A, B converge) | ∞ | 30 | **20** | 20 | | **30** | → | 20 | 20 | |
| 0 | ∞ | 30 | **20** | 20 | ∞ | **15** | | 20 | 15 | 15 |
| 1 | ∞ | 15 | **25** | 15 | ∞ | **15** | | 25 | 15 | 15 |

Figure 3: **Flow C joins at time 0, and synchronous PERC takes two iterations to converge. For simplicity, an iteration takes 1 timestep.**

shares of the previous iteration using Equation (1), and subsequently the links update the fair shares using the new demands according to Equation (2). Each iteration takes one RTT, which we define as twice the longest propagation delay for any flow in the network.

Evidently, if link $l$ is the only link in the network, $f_l$ will converge to $C_l/N$, where $N$ is the number of flows, and it will reach this fair-share rate after one iteration. However, the situation is more complicated when multiple links are involved. An example is shown in Figure 3, where flow C joins the network after flows A and B have converged. At time 0, in the first iteration of PERC, link 2 throttles the sending rate of the multi-hop flow B.This changes flow B's demand to link 1 at the beginning of the second iteration, where flow A can now use a higher fraction of the bandwidth.

Notice that the bottleneck link for flow B shifted from link 1 to link 2 in the above example, affecting fair-share rates across different iterations and extending convergence time. We call this a *dependency chain*; as flows are added or deleted and advertised fair-share rates are adjusted, bottleneck links for flows may change, which may in turn affect other bottleneck links, and so on, potentially weaving through all links in the network.

## 3.2 Asynchronous PERC

In the synchronous version of PERC, each link waits to get all updated flow demands before calculating the next fair share; conversely, a flow waits to get all updated fair share rates before it decides its sending rate. However, in a real topology, two flows may cross different links and therefore have different propagation delays; thus, with a synchronous algorithm, low-latency flows may have to suffer delays at the expense of other slower or multi-hop flows. Furthermore, flows may want to join and leave in-between a PERC *iteration* . Thus some links may be underutilized or oversubscribed during this time.

In this paper we therefore implement an *asynchronous* version of PERC, in which links update their advertised $f_l$ as soon as a new demand arrives. Also, each flow sets its sending rate[3] and demands using the most recently received set of advertised fair share rates.

[3] Flows that must increase their sending rate wait 2 RTTs for other flows to first decrease their rates and make capacity available.

A second practical consideration is calculating rates for mice flows. Short flows will require much less than their fair share rate to send all the traffic. One solution is to let flows advertise a demand that conveys the maximum rate they require over the next RTT (i.e., their size divided by the RTT). Thus, all links will assume that the short flows are bottlenecked elsewhere when calculating their advertised fair-share rates. Hence, if these flows end within one RTT, the links will not have sacrificed more bandwidth than necessary to other longer flows.

# 4. EVALUATION

Our evaluations are based on OMNET++ [26] implementations of reactive (RCP [12]), proactive (PERC, CHARNY [8]), and ideal schemes for max-min rate allocation, and ns2 [15] simulations of DCTCP [1]. We evaluate three main claims: (1) Slow convergence times lead to long FCTs, especially with higher speeds and round-trip delays; (2) PERC converges faster than Charny's state of the art proactive congestion control algorithm introduced in 90s, and therefore has smaller FCTs at higher link speeds; (3) All schemes take longer to converge when "dependency chains" between flows get longer; but reactive schemes fare much worse than proactive schemes. We now evaluate each claim in turn.

## 4.1 Slow convergence times lead to long FCTs

We test our claim by comparing FCTs of reactive vs proactive schemes for various flow sizes on a single link. We explore the effect of higher link capacities and longer RTTs.

**Setup:** We use the same workload based on a Microsoft web search datacenter cluster used in prior work [1, 3]. Flows arrive according to a Poisson process and the flow size is chosen based on empirically observed distributions. The workload has a diverse mix of small (1-10KB), medium (10KB-1MB), and large (1MB-100MB) flows. By this definition, 14% of the flows are small, 56% are medium, and 30% are large. We generate the workload over a simple dumbbell topology with a single bottleneck link. We present results for an average load of 60%; the results at other loads are qualitatively similar.

**Schemes compared:** We compare the FCT for small, medium, and large flows for reactive (RCP [12], DCTCP [1]) and proactive (CHARNY [8], PERC) congestion control algorithms. We normalize the results for each algorithm with an IDEAL reference algorithm in which all flows always transmit at the correct max-min rate. The normalization gives us a common benchmark and lets us see how close different schemes are to the ideal max-min FCT. In all of our experiments, RCP uses a default initial advertised rate of 0.5 times the link capacity, and the $\alpha$ and $\beta$ parameters for the rate update are set to 0.5 each (as suggested in [12]).

*Results and Analysis*

*PERC performs reasonably for moderate link speeds and round-trip times:* With a 10G link speed and 12$\mu s$ RTT (Fig-

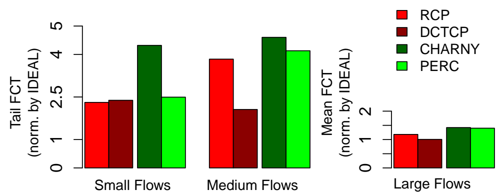



Figure 4: **Spread FCTs on 10 Gb/s link with 60% load and 12 $\mu s$ RTT: $99^{\text{th}}$ percentile for small ($< 10$ KB) and medium ($10 - 1000$ KB) flows, and mean for large ($> 1$ MB) flows. Normalized by respective statistics for ideal instantaneous max-min rate allocation.**

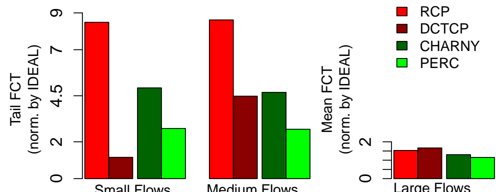



Figure 5: **Spread FCTs on 100 Gb/s link with 60% load and 12 $\mu s$ RTT: $99^{\text{th}}$ percentile for small ($< 10$ KB) and medium ($10 - 1000$ KB) flows, and mean for large ($> 1$ MB) flows. Normalized by respective statistics for ideal instantaneous max-min rate allocation.**

ure 4), PERC's FCT is comparable to that of RCP across all flow sizes. DCTCP performs better than RCP and PERC in this case, but its performance degrades significantly at higher link speeds and round-trip delays (see below).

*PERC performs much better than RCP/DCTCP at higher link speeds:* If we increase link speed to 100G (Figure 5) PERC outperforms the reactive algorithms, keeping the tail FCT very low for medium flows. Because flows can be sent much faster, medium flows that previously took 60 RTTs now finish in as few as 6 RTTs. PERC quickly converges to the optimal rates within the shorter lifetime of flows, but the reactive algorithms have a much harder time. For some flow rates DCTCP never reaches the fair share and this shows up in the long tail FCT. For RCP, all flows get the same rate, but it might be too low or too high (causing long queues).

*The difference in tail FCT between PERC and reactive algorithms is even more significant at higher RTTs:* At higher RTTs (Figure 1), reactive algorithms react more slowly since the time from measurement to reaction increases. It takes more steps to converge, and FCT grows. Note that the large bandwidth-delay product at 100G and 120$\mu s$ RTT causes DCTCP to completely break down because the ramp-up time for medium and large flows becomes prohibitively slow.

## 4.2 PERC converges faster than reactive and state of the art proactive schemes

**Setup:** We consider a topology with one TOR connected to four hosts. All 8 links, i.e., one up-link and one down-link per host, have capacity 100G each. The RTT between hosts is 12$\mu s$. We start with 32 flows between random pairs of hosts, and then alternately add a new random flow or remove an old one (randomly) after the transmission rates for the old set of flows have converged.

**Metric:** We define convergence as the first time when each of the flow rates have been within 10% of the optimal value for at least 10 consecutive RTTs. We look at the number of RTTs to converge for each of 3000 such flow changes.

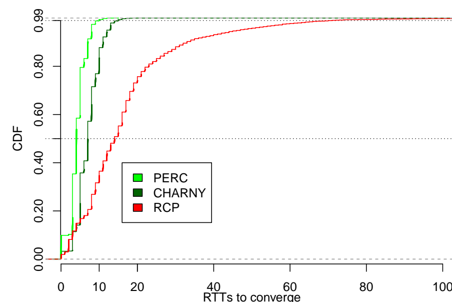



Figure 6: **CDF of convergence times for PERC, Charny's algorithm and RCP on an 8-link topology with 100G links. RTT is 12$\mu s$.**

**Schemes compared:** We compare the CDFs of convergence times over all flow changes for RCP and the two proactive schemes PERC and CHARNY in Figure 6.

*Results and Analysis*

*PERC converges faster than other proactive algorithms*: With PERC, more than 99% of the flow changes take less than 10 RTTs to converge, whereas with CHARNY convergence takes 50% longer. The median shows a similar trend. We think PERC converges faster because in the PERC algorithm, control messages carrying demands from flows to links carry more information. In CHARNY, control messages contain a single stamped rate for all links, which is updated by a link if its fair share is smaller. On the other hand, in PERC, a flow expresses a different demand for *each link*, which is the maximum it can send based on what it has heard from *all other links.*

*Proactive schemes converge faster than reactive schemes*: The median convergence time for RCP is 14 RTTs, which is twice that of CHARNY and more than 3x that of PERC. The 99th percentile convergence times is comparatively even worse at 71 RTTs, which is more than 5x that of CHARNY and 7x that of PERC. RCP's tail performance is particularly poor if there are long dependency chains, as we show in the next example.

## 4.3 Convergence times of reactive schemes scale poorly with long dependency chains

While proactive schemes scale linearly with the length of dependency chains [8], our initial simulations suggest that reactive schemes are much worse.

For example, consider the three bottleneck-link scenario shown in Figure 7. Links 1-3 are each shared by two flows, of which one is bottlenecked at a later link (except flow D). There is a dependency chain that spans three links. Figure 8 shows the time-series of the sending rates of the four flows for PERC, RCP and IDEAL.

PERC converges to the correct rates for all flows within 100$\mu s$. At around 50 $\mu s$, flow C converges to 5 Gb/s at the bottleneck, then at 100 $\mu s$, flows A and B use up all the spare capacity on the 30 Gb/s and 60 Gb/s links. As the dependency chains grow, the bottleneck information takes longer

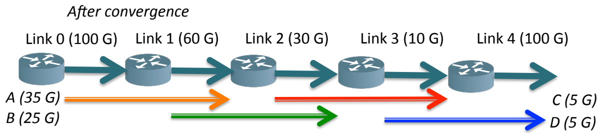



Figure 7: **A dependency chain of three. A 60 Gb/s link is shared by flows A (orange) and B (green); a 30 Gb/s link by flows B (green) and C (red); a 10 Gb/s link by flows C (red) and D (blue). RTT is 24$\mu$s.**

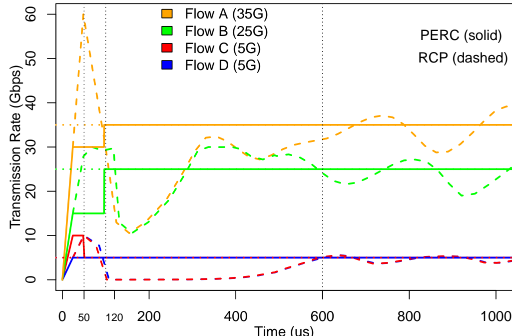



Figure 8: **Time series of flow transmission rates in 3-level-bottleneck scenario, for RCP (dashed), PERC (solid) and IDEAL (dotted).**

to propagate, growing lineraly with chain length.

RCP takes much longer to converge: flows C and D take almost 600$\mu$s to converge, and flows A and B oscillate within 20% of the target rates for milliseconds (Figure 8 only shows up to 1 ms). The bottleneck information takes time to propagate in RCP too but the convergence behavior is much worse with longer dependency chains.

## 5. DISCUSSION

**Enhancements to PERC:** In the basic message passing algorithm, each flow $a$ advertises a demand $d_l^a$ for *each* link $l \in P(a)$ on its path in the packet headers. Fortunately, a demand per hop is not necessary. Recall that the demand for a particular link is computed as: $d_l^a = \min_{l' \in P(a), l' \neq l}\{f_{l'}\}$, where $(f_l : l \in P(a))$ are the fair share rates fed back by the links on flow $a$'s path. We observe that the demands take only one of two values: either the minimum or the second minimum of the fair shares. Specifically, if link $l^*$ is the bottleneck link on path $P(a)$ ($f_{l^*}$ is the smallest fair share), then $d_l = f_{l^*}$ for any link $l \neq l^*$; and $d_{l^*} = \min_{l' \in P(a), l' \neq l^*}\{f_{l'}\}$ (second minimum). This suggests a simple optimization where the packet header carries only two demand values corresponding to the minimum and second minimum fair share values.

Similarly, sending one control packet every RTT for each flow can become untenable. For example, for a 3.2$\mu$s RTT, a 40B packet each RTT corresponds to 100 Mb/s of control traffic per flow. The control overhead can be reduced by piggybacking and only communicating values that change. Note that control overhead is less important at higher speeds.
**Implementing PERC in programmable switches:** PERC requires switches to compute their fair shares explicitly from the advertised flow demands. The algorithm will need further simplifications to be practical. Nonetheless, flexible architectures such as Reconfigurable Match Action (RMT) [7] and switch programming languages such as P4 [6] provide avenues for PERC to be deployed.
**Centralized Implementation:** Proactive congestion control could be implemented using a central rate allocator. Upon each change (flow start or end), the central allocator would calculate all sending rates in accordance with a rate allocation policy (e.g., max-min fairness, earliest deadline first, etc) and send the allocated rates to all affected flow. The central implementation is conceptually simple and can support arbitrary allocation policies but has challenges around scalability and responsiveness. The central allocator may need to scale to millions of flows with microsecond response times. Yet, a central rate allocator would be more scalable than Fastpass [22], a recently proposed system that advocates central control for every packet transmission (instead of every flow).

## 6. RELATED WORK

The tradeoff between convergence speed and stability has long been recognized in the congestion control literature [18, 13]. XCP [18] and RCP [12] replaced TCP's window size adaptation with a more direct calculation at the switch to achieve the right rates more rapidly. However, as we have shown, these schemes are still rather slow and require many round-trips to converge. Kelly [19] was the first to recognize that congestion control can be thought of as solving an optimization problem for resource allocation; this viewpoint makes precise the intuition that reactive congestion control protocols like TCP and RCP are distributed implementations of a gradient descent procedure.

Many datacenter transports are optimized for different allocation objectives, e.g., network latency [2, 1], bandwidth guarantees [4, 17] , deadlines [27, 25], flow completion time [3, 14], and coflow completion times [10, 11]. While we focus on max-min policy in this paper (the simplest objective most general purpose congestion control objectives strive for), we expect that our main observations regarding reactive vs proactive schemes to apply to other objectives.

## 7. CONCLUSIONS

This paper makes three points: convergence time will be an important metric for congestion control schemes as we scale to higher speeds, proactive congestion control algorithms can provide fast convergence times, and message passing algorithms like PERC can outperform traditional explicit rate control algorithms. We also argue that faster convergence times impact higher level metrics such as the tail latency of medium and small flows. Given the new milieu of programmable networks, we believe that proactive algorithms like PERC are both needed and possible to deploy, at least within high speed data centers.

## Acknowledgements

This work was supported by National Science Foundation (NSF) grant CNS-1040593 and NSF Graduate Research Fellowship Program grant DGE-114747.

## 8. REFERENCES


[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proc. ACM SIGCOMM 2010 Conference*, pages 63–74, Aug 2010.

[2] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is more: trading a little bandwidth for ultra-low latency in the data center. In *Proc. USENIX Conference on Networked Systems Design and Implementation (NSDI 2012)*, April 2012.

[3] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pFabric: minimal near-optimal datacenter transport. In *Proc. ACM SIGCOMM 2013 Conference*, pages 435–446, Aug 2013.

[4] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstrong. Towards predictable datacenter networks. In *Proc. ACM SIGCOMM 2011 Conference*, pages 242–253, Aug 2011.

[5] D. Bertsekas and R. Gallager. *Data Networks*, pages 524–527. Prentice Hall, 2nd edition, 1992.

[6] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Computer Communication Review*, pages 87–95, Jul 2014.

[7] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN. In *Proc. ACM SIGCOMM 2013 Conference*, pages 99–110, Aug 2013.

[8] A. Charny, D. D. Clark, and R. Jain. Congestion control with explicit rate indication. In *Proc. IEEE International Conference on Communications (ICC)*, 1995.

[9] M. Chiang, S. H. Low, J. C. Doyle, et al. Layering as optimization decomposition: A mathematical theory of network architectures. *Proceedings of the IEEE*, 95(1):255–312, 2007.

[10] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *Proc. ACM SIGCOMM 2014 Conference*, pages 443–454, Aug 2014.

[11] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. Decentralized task-aware scheduling for data center networks. In *Proc. ACM SIGCOMM 2014 Conference*, pages 431–442, Aug 2014.

[12] N. Dukkipati. *Rate Control Protocol (RCP): Congestion control to make flows complete quickly*. PhD thesis, Citeseer, 2007.

[13] C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong. A control theoretic analysis of red. In *Proc. IEEE INFOCOM*, pages 1510–1519, Apr 2001.

[14] C.-Y. Hong, M. Caesar, and P. B. Godfrey. Finishing flows quickly with preemptive scheduling. In *Proc. ACM SIGCOMM 2012 Conference*, pages 127–138, Aug 2012.

[15] T. Issariyakul and E. Hossain. *Introduction to Network Simulator NS2*. Springer Publishing Company, 1st edition, 2008.

[16] V. Jacobson and M. J. Karels. Congestion avoidance and control. In *Proc. ACM SIGCOMM 1988 conference*, Aug. 1988.

[17] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, A. Greenberg, and C. Kim. EyeQ: Practical network performance isolation at the edge. In *Proc. USENIX Conference on Networked Systems Design and Implementation (NSDI 2013)*, Apr 2013.

[18] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proc. ACM SIGCOMM 2002 Conference*, Aug 2002.

[19] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control for communication networks: shadow prices, proportional fairness, and stability. In *The Journal of the Operational Research Society*, pages 237–252, Mar 1998.

[20] S. H. Low and L. L. H. Andrew. Understanding xcp: equilibrium and fairness. In *Proc. IEEE INFOCOM*, pages 1025–1036, Mar 2005.

[21] D. J. MacKay. *Information theory, inference and learning algorithms*. Cambridge University Press, 2003.

[22] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: a centralized "zero-queue " datacenter network. In *Proc. ACM SIGCOMM 2014 Conference*, pages 307–318, Aug 2014.

[23] T. J. Richardson and R. L. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. In *Proc. IEEE Transactions on Information Theory*, pages 599–618, Feb 2001.

[24] R. Srikant. *The mathematics of Internet congestion control*. Springer Science & Business Media, 2012.

[25] B. Vamanan, J. Hasan, and T. N. Vijaykumar. Deadline-aware datacenter TCP (D2TCP). In *Proc. ACM SIGCOMM 2012 Conference*, pages 115–126, Aug 2012.

[26] A. Varga and R. Hornig. An overview of the OMNeT++ simulation environment. In *Proc. SIMUTools*, Mar 2008.

[27] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: meeting deadlines in datacenter networks. In *Proc. ACM SIGCOMM 2011 Conference*, pages 50–61, Aug 2011.