CONSISTENT HIGH PERFORMANCE AND FLEXIBLE CONGESTION CONTROL ARCHITECTURE

BY

MO DONG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Doctoral Committee:

Associate Professor P. Brighten Godfrey, Chair & Director of Research
Associate Professor Matthew Caesar
Professor Klara Nahrstedt
Dr. Jana Iyengar, Google

Abstract

The part of TCP software stack that controls how fast a data sender transfers packets is usually referred as *congestion control*, because it was originally introduced to avoid network congestion of multiple competing flows. During the recent 30 years of Internet evolution, traditional TCP congestion control architecture, though having a army of specially-engineered implementations and improvements over the original software, suffers increasingly more from surprisingly poor performance in today's complicated network conditions. We argue the traditional TCP congestion control family has little hope of achieving consistent high performance due to a fundamental architectural deficiency: hardwiring packet-level events to control responses.

In this thesis, we propose Performance-oriented Congestion Control (PCC), a new congestion control architecture in which each sender continuously observes the connection between its rate control actions and empirically experienced performance, enabling it to use intelligent control algorithms to consistently adopt actions that result in high performance. We first build the above foundation of PCC architecture analytically prove the viability of this new congestion control architecture. Specifically, we show that, controversial to intuition, with certain form of utility function and a theoretically simplified rate control algorithm, selfishly competing senders converge to a fair and stable Nash Equilibrium. With this architectural and theoretical guideline, we then design and implement the first congestion control protocol in PCC family: PCC Allegro. PCC Allegro immediate demonstrates its architectural benefits with significant, often more than 10×, performance gain on a wide spectrum of challenging network conditions. With these very encouraging performance validation, we further advance PCC's architecture on both utilty function framework and the learning rate control algorithm. Taking a principled approach using online learning theory, we designed PCC Vivace with a new strictly socially concave utility function framework and a gradient-ascend based learning rate control algorithm. PCC Vivace significantly improves performance on fast-changing networks, yields better tradeoff in convergence speed and stability and better TCP friendliness comparing to PCC Allegro and other state-of-art new congestion control protocols. Moreover, PCC Vivace's expressive utility function framework can be tuned differently at different competing flows to produce predictable converged throughput ratios for each flow. This opens significant future potential for PCC Vivace in centrally control networking paradigm like Software Defined Networks (SDN). Finally, with all these research advances, we aim to push PCC architecture to production use with a a user-space tunneling proxy and successfully integration with Google's QUIC transport framework.

To Yuesheng and my parents.

Acknowledgments

Looking back these five years, I had an unexpectedly exciting journey. For that, I would like to express my deepest gratitude to my adviser Brighten Godfrey. He taught me to identify and work on important research challenges, provided me numerous brilliant technical insights in our weekly meetings and supported me with firm trust through the difficult times in the beginning of my thesis research. He has taught me how to approach research problems and cultivated me to become an independent researcher. He did not only give me high degree of freedom in research but also altruistically tailored his invaluable efforts and guidance to fit my future career goals. When I was a child, I was taught an old Chinese saying: a true mentor shall be respected as one's own father. I regard Brighten, with his pure and altruistic mentorship, as my first true mentor. I also feel fortunate to be his friend and share many fun and exciting moments in life. I would also like to sincerely thank professor Matthew Caesar. I will always remember the midnight when he came to office to find a server for a time-sensitive experiment of mine. I also received invaluable feedbacks on various research projects from him. I would also like to thank professor Klara Nahrstedt and Jana Iyengar, who provided validation in various stages of my research and served in my PhD committee later on.

I feel lucky to collaborate with some of the most brilliant minds in networking community. I would like to thank Qingxi Li, Tong Meng, Xuefeng Zhu, Michael Schapira, Doron Zarchy and Yossi Gilad for their indispensable collaboration on PCC project. I would also like to thank my friends and colleagues in NetSys group: Sangeetha A. J. and Santhosh Prabhu, for their collaboration on research projects and other fun projects that are not included in the thesis. I would like thank Ian Swett, Jo Kulik and Jana Iyengar for their interests in my thesis research and, collaboration and help to push PCC to large-scale production deployments. I would like to thank Igor Ganichev and Teemu Koponen who were my mentors during my internship in Nicira and taught me a great deal about principled engineering practices in a short period of time.

I also want to take this opportunity to thank my dear friends who made my 5 years in UIUC not only fun but sometimes very adventurous. The list can go on, so I am just writing down the first several names pop into my mind: Qingxi Li, Santhosh Prabhu, Sangeetha AJ, Guliz Tuncay, Wenxuan Zhou, Chiyao Hong, Zhenhuan Gao, Yunlong Gao, Haiming Jin, Shannon Chen, Grace Yen, Lu Su, Xuefeng Zhu, Bei Zhang, Andrew Kryczka, Faraz Faghri, Fangbo Tao, Shiyan Zhang, Xuran Peng, Lei Jin, Ran Chao, Yunfan Shi, Guangxiang Du, Shaohan Hu, Bingzhe Liu, Su Du, Meng Wei, Zhuotao Liu, Weijie Liu, Mingcheng Chen, Giri Subramanian, Debish Fesehaye, Yutong Qiu, Kana Ruan and so many more.

I also got tremendous help from many staff members in CS department, especially from Mary Beth Kelly and Andrea Whitesell.

Also, I would like to thank Internet2, National Science Foundation and Google, for their direct financial support for my research.

Most importantly, I want to thank my parents. They provided me the luxury of a warm and loving family and very high quality early education. In fact, they introduced me to science and technology and mentored my first principled research project about population structure change prediction in China when I was at high school. Their unreserved love have and will continue to motivate me moving forward and their kindness towards the world will always remind me to do the same. Lastly, I want to thank my girlfriend Yuesheng Ruan for her unwavering love and unconditional faith in me for the last seven years. Her accompany and many sacrifices created a brilliant beacon to help me sail through some very dark times in the past five years. I could not achieve this success without her. To my parents and Yuesheng, I dedicate this thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent 30 years, the world has witnessed the evolution of Internet that is far beyond anyone's imagination. The sheer scale of Internet is astonishing: recent reports [5, 9] show that 3 billion users are connected to Internet in 2015, 8.7 billion devices are Internet connected in 2014 and the number of connected devices is projected to reach 20 billion to 50 billion by 2020. On top of this astronomical scale, Internet hosts a galaxy of applications that has become essential to every parts of people's life: from social networking to remote conferencing, from video games to career development, from finding the best restaurant to election of prime minister, and the list goes on. This trend is nowhere near stopping as recent innovative applications, such as Virtual Reality and Internet of Things, flood in and continuously revolutionize our life.

Needless to say, none of these exciting evolution or applications can exist without transferring data between different endpoints across the Internet. One most commonly used Internet software stack to deliver data between any two endpoints is called Transport Control Protocol (TCP). TCP is arguably most widely used piece of software in Internet as it exists in almost all Internet connected endpoints (e.g. personal computers, mobile phones, application servers and etc.). TCP has two key functionalities: first, it provides the upper layer application with a reliable data delivery abstraction layer, i.e. "connection", that masks the unreliability of lower networking layers; second, and more importantly, it dictates *the data delivery performance*, and *the stable and fair network resource sharing* among competing data flows in the networks. The second functionality of TCP is usually referred as Congestion Control.

The first version of TCP congestion control [47] was developed around 30 years ago. Internet at that time was so much simpler and smaller comparing to what it has become today. Moreover, the computer processing capability was also much more limited. Under those historical conditions, a simple yet elegant congestion control architecture was established that controls a flow's sending rate using congestion window and the control action is triggered by individual packet-level feedbacks (e.g. ACKs) from the receiver. The first realization of this architecture uses a very simple rate control algorithm called Additive Increase Multiplicative Decrease (AIMD) [47]. As this thesis is mostly closely related to the congestion control part of TCP, we use "TCP" to refer to its traditional congestion control architecture.

As the Internet's scale explosively increases, the network conditions and environments have become significantly more complex and heterogeneous. To list a few examples: the available network bandwidth has increased more than 200×; distance of communication has changed from relatively local to global cross-continental and even into the space; less reliable WiFi networks become a pervasive last-hop connection option for end users; cellular networks covers increasingly large amount of population world-wide; hyper-scale data center networks are built to scale-out applications; a heterogeneous group of Internet Service Providers with complicated relationships are established; large amount of video traffic starts to dominates the Internet data volume; and so on. These changes lead to an immense diversity and complexity of network conditions: random loss and zero loss; shallow queues and bufferbloat; links from Kbps to Gbps; Active Queue Management (AQMs) deployed at different points of the network; software routers; rate shaping and policy at gateways; virtualization layers; middleboxes like firewalls, packet inspectors and load balancers; RTTs of competing flows can vary by more than 1000×; network condition can change in a split of second with the channel condition of wireless or cellular networks; data flow path changes can happen with centralized traffic engineering. These complexity and changes have caused severe performance problems for traditional TCP congestion control architecture. For example, TCP performs poorly on lossy links, penalizes high-RTT flows, under-utilizes high bandwidth-delay product (BDP) connections, cannot handle rapidly changing networks, can collapse under data center incast [29] and cannot efficiently utilize network bandwidth with small network buffer.

Two broad avenues of research have been pursued to improve the performance of traditional congestion control. The first is for network devices to give end-hosts explicit feedback: e.g. conveying the extent of congestion via ECN bits [21] or explicitly prescribing window sizes or rates as in XCP [52] and RCP [34]. But these designs require hardware and configuration changes and (for [34, 52]) packet header changes with support, or at least noninterference, by all devices along an end-to-end path. Due to not only the technical difficulty but also the lack of incentive for network operators to standardize and deploy these in-network solutions, this has been proved too high a bar for widespread deployment.

In the second avenue of research, end-host-based TCP modifications have addressed specific problems, resulting in an innumerably long string of designs tweaking TCP. These include using latency rather than just loss as a signal of congestion [74], sophisticated window expansion algorithms [43], coordinated adjustment of the receive window across connections at the receiver to mitigate incast [79], viewing a certain amount of packet loss as unrelated to congestion [56], expanding the window quicker by setting up a reference RTT on a satellite link [27] and so on. However, the very fact that there are such a large number of modifications indicates that each of them is only a point solution: they yield better performance in certain cases, but break in others. Although they use distinct congestion control algorithms, all these TCP variants inherit the same TCP-based congestion control architecture: hardwiring certain *predefined packet-level events* deterministically to certain *predefined control responses* which we refer to as *hardwired mapping*. A hardwired mapping has to make *assumptions* about the network. When the assumption is violated, the corresponding control action can severely degrade performance. In fact, it is fundamentally hard to formulate an “always optimal” hardwired mapping in a complex real-world network. We discuss this point in great detail in Chapter 2. Therefore, to sustainably support the complexity introduced in the rapid evolution of Internet, fundamental rethinking of the traditional TCP congestion control architecture is in dire need.

1.1 Thesis Statement

In my thesis, we address the grand challenge of redesigning the architecture of congestion control protocols to achieve consistent high performance under a wide spectrum of challenging network conditions with the additional goal of being easy to deploy and flexible to different application scenarios.

1.2 Performance-oriented Congestion Control Architecture

In Chapter 2, we propose a new congestion control architecture, called Performance-oriented Congestion Control (PCC). Different from TCP's hardwired mapping architecture, after choosing a certain data transfer rate, a PCC sender does not rigidly react to individual packet-level feedback events. Instead, it continuously aggregates these packet-level events (e.g. selective ACK) into meaningful performance metrics such as packet loss rate, goodput and latency. Then, it combines these empirically observed performance metrics into a utility value, describing its data transfer objective, using a utility function. This observation and aggregation process allows the sender to establish a casual connection between its *rate control decision* and *empirically experienced performance*. Leveraging this connection, PCC sender treats the network as a blackbox and consistently adopts actions that result in empirically higher performance without making any assumption about underlying network conditions. Intuitively, PCC rises from where TCP architecture fails and has great potential to achieve consistent high performance. Formally, we analyze the viability of this overall architectural proposal by establishing the theoretical foundation regarding convergence and stability. We prove that even though individual PCC senders are selfish in nature, it is possible to achieve global convergence and fair share of network resources leveraging the game theoretical dynamics of utility functions and a rate control algorithm based using the causal relation between control action and resulting performance. Though it is a substantial shift in architecture, PCC can be deployed by only replacing the sender-side rate control of TCP.

1.3 PCC Allegro: A Strong Case of PCC Architecture

In Chapter 3, we present the first instantiation, called PCC Allegro, of the PCC architecture. We design the mechanisms of the performance monitoring process and a performance oriented rate control algorithm. As the algorithm analyzed in Chapter 1 is oversimplified for theoretical proof purposes, we design a working rate control algorithm by addressing many important challenges regarding the speed to converge to optimal sending rate and the robustness of the rate control algorithm in the face of noisy measurements. With achieving consistent high performance as PCC's key goal, we experimentally evaluated a real-world implementation of PCC Allegro against a wide range of challenging network conditions. PCC's architecture shows significant advantages over existing traditional TCP congestion control architecture. PCC often beats *specially engineered* TCPs in various network environments by more than 10X: **(a.)** in the wild on the global commercial Internet (often more than **10×** the throughput of TCP CUBIC); **(b.)** inter-data center networks (**5.23×** vs. TCP Illinois); **(c.)** emulated satellite Internet links (**17×** vs TCP Hybla); **(d.)** unreliable lossy links (**10 − 37×** vs Illinois); **(e.)** unequal RTT of competing senders (an **architectural cure** to RTT unfairness); **(f.)** shallow buffered bottleneck links (up to **45×** higher performance, or **13×** less buffer to reach 90% throughput); **(g.)** rapidly changing networks (**14×** vs CUBIC, **5.6×** vs Illinois). In addition to its consistent high performance, we show that PCC Allegro achieves significantly better convergence stability and reactivity trade off comparing to TCP family and has potential of expressing different data transfer objectives with its pluggable utility function.

1.4 PCC Vivace: Online Learning Congestion Control

In Chapter 4, we take a step back and review the two essential architectural components of PCC: utility function framework and learning based rate control algorithm. We identify the design of these two components as design of a online learning system. Therefore, we leverage design principles and state-of-art theoretical analysis tools from the rich literatures on *online optimization* in machine learning to design PCC Vivace. For the learning rate control component, PCC Vivace employs a no-regret gradient-ascent-based online optimization algorithm to achieve quick utilization of spare network capacity, swift reaction to network changes, and fast and stable convergence. On the utility function framework front, PCC Vivace relies on a new, learning-theory-informed framework for utility derivation that incorporates crucial considerations such as reducing self-inflicted latency and TCP friendliness. This new utility function framework also exposes a fundamental tradeoff, seemingly applicable beyond PCC's architecture, between random loss tolerance and packet loss at convergence. We turn the above theoretical insights into a operational system by overcoming major engineering challenges such as: (1) transforming online optimization algorithms from machine learning literature into practical rate-control schemes by striking a delicate balance between reactivity and stability; and (2) gathering useful statistics about performance via online measurements in the face of very limited measurement time, changing network conditions, noisy network feedback, and beyond. Extensive experimentation with PCC Vivace, BBR, PCC Allegro, and TCP in controlled environments, real residential Internet scenarios, and with video-streaming applications, demonstrates that PCC Vivace significantly improves upon all other evaluated congestion control schemes. Beyond the merits above, PCC Vivace unleashes potential of PCC's expressive utility function framework: competing flows can be tuned to use different utility functions to produce predictable convergence throughput ratios for each flow. This can enable centralized network control (SDN or OpenTCP [40]) to allocate network capacity intelligently.

1.5 Pushing PCC to Real-world Deployments

In Chapter 5, we seek to push PCC to large-scale and real-world deployments to benefit the global Internet's transport layer performance. In order to reach that ambitious goal, we explore the following two technical directions. First, we build a PCC tunneling proxy using our current user space implementation. This proxy can tunnel TCP traffic between a pair of sender and receiver through PCC's high performance transport to boost applications' performance. Using this proxy, we demonstrate PCC can immediately benefit existing applications such as video streaming and cross-continental file transferring. This opens up huge application potentials for PCC in WAN optimization and acceleration scenarios. Second, we explore the path to integrate PCC with a widely used user-space congestion control and transport framework, QUIC [15] from Google. We demonstrate PCC's deployability by implementing PCC as a congestion control module in QUIC and replicated some of experiments demonstrating the viability and performance parity of QUIC implementation. We also successfully deploy PCC congestion control QUIC module in production infrastructure in Google and preliminarily tested its performance comparing to the CUBIC congestion control module in QUIC.

1.6 Dissertation Plan

This thesis follows the above flow in the following four chapters. And in chapter 6, we conclude the thesis, lay out the unresolved challenges, future research directions and deployments roadmaps.

Chapter 2

Performance-oriented Congestion Control Architecture

2.1 Introduction

In the roughly 30 years since its deployment, TCP's congestion control architecture has been notorious for degraded performance. TCP performs poorly on lossy links, penalizes high-RTT flows, underutilizes high bandwidth-delay product (BDP) connections, cannot handle rapidly changing networks, can collapse under data center incast [29] and incurs very high latency with bufferbloat [39] in the network.

As severe performance problems have accumulated over time, protocol "patches" have addressed problems in specific network conditions such as high BDP links [43, 74], satellite links [27, 63], data center [21, 79], wireless and lossy links [56, 57], and more. However, the fact that there are so many TCP variants suggests that each is only a point solution: they yield better performance under specific network conditions, but break in others. Worse, we found through real-world experiments that in many cases these TCP variants' performance is *still far away from optimal even in the network conditions for which they are specially engineered.* Indeed, TCP's low performance has impacted industry to the extent that there is a lucrative market for special-purpose high performance data transfer services [1, 2, 13, 16].

Thus, the core problem remains largely unsolved: *achieving consistently high performance over complex real-world network conditions.* We argue this is indeed a very difficult task within TCP's rate control architecture, which we refer to as *hardwired mapping*: certain predefined packet-level events are hardwired to certain predefined control responses. TCP reacts to events that can be as simple as "one packet loss" (TCP New Reno) or can involve multiple signals like "one packet loss and RTT increased by $x\%$" (TCP Illinois). Similarly, the control response might be "halve the rate" (New Reno) or a more complex action like "reduce the window size w to $f(\Delta RTT)w$"

(Illinois). The defining feature is that the control action is a direct function of packet-level events.

A hardwired mapping has to make *assumptions* about the network. Take a textbook event-control pair: a packet loss halves the congestion window. TCP *assumes* that the loss indicates congestion in the network. When the assumption is violated, halving the window size can severely degrade performance (e.g. if loss is random, rate should stay the same or increase). It is fundamentally hard to formulate an "always optimal" hardwired mapping in a complex real-world network because the actual optimal response to an event like a loss (i.e. decrease rate or increase? by how much?) is sensitive to network conditions. And modern networks have an immense diversity of conditions: random loss and zero loss, shallow queues and bufferbloat, RTTs of competing flows varying by more than 1000×, dynamics due to mobile wireless or path changes, links from Kbps to Gbps, AQMs, software routers, rate shaping at gateways, virtualization layers and middleboxes like firewalls, packet inspectors and load balancers. These factors add complexity far beyond what can be summarized by the relatively simplistic assumptions embedded in a hardwired mapping. Most unfortunately, when its assumptions are violated, TCP still rigidly carries out the harmful control action.

In this chapter, we propose a new congestion control architecture: Performance-oriented Congestion Control (PCC). PCC's goal is to understand what rate control actions improve performance based on *live experimental evidence*, avoiding TCP's assumptions about the network. PCC sends at a rate r for a short period of time, and observes the results (e.g. SACKs indicating delivery, loss, and latency of each packet). It aggregates these packet-level events into a utility function that describes an objective like "high throughput and low loss rate". The result is a single numerical performance utility u. At this point, PCC has run a single "micro-experiment" that showed sending at rate r produced utility u. To make a rate control decision, PCC runs multiple such micro-experiments: it tries sending at two different rates, and moves in the direction that empirically results in greater performance utility. This is effectively *A/B testing for rate control* and is the core of PCC's decisions. PCC runs these micro-experiments continuously (on every byte of data, not on occasional probes), driven by an online learning algorithm that tracks the empirically-optimal sending rate. Thus, rather than making assumptions about the potentially-complex network, PCC adopts the actions that *empirically* achieve consistent high performance. PCC's rate control is selfish in nature, but surprisingly, using a widely applicable utility function, competing PCC senders provably converge to a fair equilibrium (with a single bottleneck link). In this chapter, we analytically prove the above statement and therefore establish the viability of PCC architecture. Moreover, the ability to express different objectives via choice of the utility function (e.g. throughput or latency) provides a flexibility beyond TCP's architecture.

2.2 The Key Idea of PCC Architecture

Suppose flow f is sending a stream of data at some rate and a packet is lost. How should f react? Should it slow the sending rate, or increase, and by how much? Or leave the rate unchanged? This is a difficult question to answer because real networks are complex: a single loss might be the result of *many* possible underlying network scenarios. To pick a few:

- f may be responsible for most of congestion. Then, it should decrease its rate.
- f might traverse a shallow buffer on a high-BDP link, with the loss due to bad luck in statistical multiplexing rather than high link utilization. Then, backing off a little is sufficient.
- There may be a higher-rate competing flow. Then, f should maintain its rate and let the other back off.
- There may be random non-congestion loss somewhere along the path. Then, f should maintain or increase its rate.

Classically, TCP assumes a packet loss indicates non-negligible congestion, and that halving its rate will improve network conditions. However, this assumption is false and will degrade performance in three of the four scenarios above. Fundamentally, picking an optimal *predefined and hardwired* control response is hard because for the same packet-level events, a control response optimal under one network scenario can decimate performance in even a slightly different scenario. The approach taken by a large number of TCP variants is to use more sophisticated packet-level events and control actions. But this does not solve the fundamental problem, because the approach *still hardwires predetermined events to predetermined control responses*, thus inevitably embedding unreliable assumptions about the network. When the unreliable assumptions are violated by the

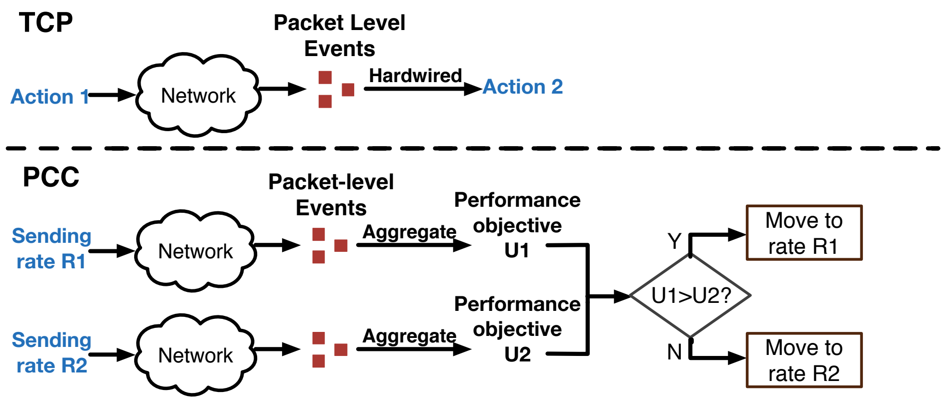


Figure 2.1: The decision-making structure of TCP and PCC.

complexity of the network, performance degrades severely. For example, TCP Illinois [56] uses both loss and delay to form an event-control mapping, but its throughput collapses with even a small amount of random loss, or when network conditions are dynamic. More examples are in §2.5.

Most unfortunately, if some control actions are indeed harming performance, TCP can still blindly "jump off the cliff", because it does not notice the control action's actual effect on performance.

But that observation points toward a solution. Can we design a control algorithm that directly understands whether or not its actions actually improve performance?

Conceptually, no matter how complex the network is, if a sender can directly measure that rate r_1 results in better performance than rate r_2, it has some evidence that r_1 is better than sending at r_2 — at least for this one sender. This example illustrates the key design rationale behind **Performance-oriented Congestion Control** (PCC): PCC makes control decisions based on *empirical evidence pairing* ***actions*** *with directly* ***observed performance*** *results.*

PCC's control action is its choice of sending rate. PCC divides time into continuous time periods, called *monitor intervals* (MIs), whose length is normally one to two RTTs. In each MI, PCC tests an action: it picks a sending rate, say r, and sends data at rate r through the interval. After about an RTT, the sender will see selective ACKs (SACK) from the receiver, just like TCP. However, it does not trigger any predefined control response. Instead, PCC aggregates these SACKs into meaningful performance metrics including throughput, loss rate and latency. These performance metrics are combined to a numerical utility value, say u, via a *utility function*. The utility function can be customized for different data transmission objectives, but for now the reader can assume the objective of "high throughput and low loss rate", such as $u = T - L$ (where $T =$ throughput and $L =$ loss rate) which will capture the main insights of PCC. The end result is that PCC knows when it sent at rate r, it got utility of u.

The preceding describes a single "micro-experiment" through which PCC associates a specific *action* with an observed *resulting utility*. PCC runs these micro-experiments continuously, comparing the utility of different sending rates so it can track the optimal action over time. More specifically, PCC runs an online learning algorithm similar to gradient ascent. When starting at rate r, it tests rate $(1+\varepsilon)r$ and rate $(1-\varepsilon)r$, and moves in the direction (higher or lower rate) that empirically yields higher utility. It continues in this direction as long as utility continues increasing. If utility falls, it returns to a decision-making state where it again tests both higher and lower rates to find which produces higher utility.

Note that PCC does not send occasional probes or use throwaway data for measurements. It observes the results of its actual control decisions on the application's real data and does not pause sending to wait for results.

We now return to the example of the beginning of this section. Suppose PCC is testing rate 100 Mbps in a particular interval, and will test 105 Mbps in the following interval. If it encounters a packet loss in the first interval, will PCC increase or decrease? In fact, there is no specific event in a single interval that will always cause PCC to increase or decrease its rate. Instead, PCC will calculate the utility value for each of these two intervals, and move in the direction of higher utility. For example:

- If the network is congested as a result of this flow, then it is likely that sending at 100 Mbps will have similar throughput and lower loss rate, resulting in higher utility. PCC will decrease its rate.
- If the network is experiencing random loss, PCC is likely to find that the period with rate 105 Mbps has similar loss rate and slightly higher throughput, resulting in higher utility. PCC will therefore increase its rate despite the packet loss.

Throughout this process, PCC makes no assumptions about the underlying network conditions, instead observing which actions empirically produce higher utility and therefore achieving consistent high performance. **Many issues remain.** We next delve into fairness, convergence, and choice of utility function.

2.3 Fairness and Convergence

Each PCC sender optimizes its utility function value based only on locally observed performance metrics. However, this local selfishness does not imply loss of global stability, convergence and fairness. We next show that when selfish senders use a particular "safe" utility function and a simple control algorithm, they provably converge to fair rate equilibrium.

We assume n PCC senders $1, \ldots, n$ send traffic across a bottleneck link of capacity $C > 0$. Each sender i chooses its sending rate x_i to optimize its utility function u_i. We choose a utility function expressing the common application-level goal of "high throughput and low loss":

$$u_i(x_i) = T_i \cdot Sigmoid_\alpha(L_i - 0.05) - x_i \cdot L_i$$

where x_i is sender i's sending rate, L_i is the observed data loss rate, $T_i = x_i(1 - L_i)$ is sender i's throughput, and $Sigmoid_\alpha(y) = \frac{1}{1+e^{\alpha y}}$ for some $\alpha > 0$ to be chosen later.

The above utility function is derived from a simpler starting point: $u_i(x_i) = T_i - x_i \cdot L_i$, i.e., i's throughput minus the production of its loss rate and sending rate. However, we observed that this utility function will make loss rate at equilibrium point approach 50% when the number of competing senders increases. Therefore, we include the sigmoid function as a "cut-off". When α is "big enough", $Sigmoid_\alpha(L_i - 0.05)$ will rapidly get closer to 0 as soon as L_i exceeds 0.05, leading to a negative utility for the sender. Thus, we are setting a barrier that caps the overall loss rate at approximately 5% in the worst case.

To analyze the equilibrium of this Game theoretic system, we assume a FIFO queue single bottleneck scenario. Under this assumption, the observed packet loss rate can be expressed as $L(x) = \max\{0, 1 - \frac{C}{\Sigma_j x_j}\}$. Without loss of generality, we set the capacity link to C=1 in the following analysis. Also, let $S(x) \triangleq \sum_{j \in N} x_j$, $Y(x) \triangleq e^{a(0.95 - \frac{1}{S(x)})}$, and $Sig(x) \triangleq \frac{1}{1+Y(x)}$.

2.3.1 Unique Fair Stable State

We show here that this careful choice of utility function to embed into PCC induces a unique stable state of sending rates, that is, a state of sending rates $x^* = (x_1^*, \ldots, x_n^*)$ such that no single sender can improve its performance by changing its sending rate. In game-theoretic terminology, this translates to the existence of a unique Nash equilibrium.

Theorem 1. *When $n \geq 3$ and $\alpha \geq \max\{2.2(n-1), 100\}$, there exists a unique stable state of sending rates $x_1, \ldots, x_n$.*

Our proof relies on analyzing three possible cases: (1) the sum of sending rates is less than the link capacity, that is, $\Sigma_j x_j < 1$; (2) $1 \leq \Sigma_j x_j \leq \frac{20}{19}$; and (3) $\Sigma_j x_j > \frac{20}{19}$. We first show that there can only exists a stable state in case (2). We then show that there is a unique stable state in case (2).

We present the following simple claim, which pertains to the first of these three cases:

Claim 1. *Let $n \geq 3$ and $\alpha \geq max(2.2(n-1), 100)$. Any configuration of seding rates $x = (x_1, \ldots, x_n)$ such that $\Sigma_j x_j < 1$ is not a stable state.*

Proof. Consider a specific sender i, observe that when the sum of sending rates is strictly less than the link capacity, then according to the definition of i's utility function i can achieve better performance by increasing its transmission rate without exceeding the capacity. □

Claim 2. *Let $n \geq 3$ and $\alpha \geq max(2.2(n-1), 100)$. Any configuration of seding rates $x = (x_1, \ldots, x_n)$ such that $\Sigma_j x_j > \frac{20}{19}$ is not a stable state.*

Proof. The partial derivative of each sender i, when fixing the other senders' rates is:

$$\frac{\partial u_i(x)}{\partial x_i} = \frac{Sig(x)+1}{S(x)} - \frac{x_i(Sig(x)+1)}{S(x)^2} - \frac{\alpha x_i Y(x) Sig^2(x)}{S(x)^3} - 1 \tag{2.1}$$

Let $\Phi(x) \triangleq \sum_{i \in N} \frac{\partial u_i(x)}{\partial x_i}$. Summing the utilities over all senders we get,

$$\Phi(x) = \frac{(n-1)(Sig(x)+1)}{S(x)} - \frac{\alpha Y(x)}{S(x)^2} Sig^2(x) - n \tag{2.2}$$

Since $\Phi(x)$ depends only on $S(x)$, we abuse notation and denote eq. 2.2 by $\Phi(s)$ where $s = S(x)$.

Let $s = \Sigma_j x_j$. By (2.2), if $s > \frac{20}{19}$ then $\Phi(s) < 0$. Therefore, there must be at least one sender i with $\frac{\partial u_i(x)}{\partial x_i} < 0$ and hence x is not a stable state. $\square$

Lemma 1. *Let $n \geq 3$ and $\alpha \geq max(2.2(n-1), 100)$. Then, there exists a stable state x such that $\Sigma_j x_j \in (1, 20/19)$.*

Proof. By (2.2) we get

$$\Phi(1) = \frac{n-1}{1+e^{-0.05\alpha}} - \frac{\alpha e^{-0.05\alpha}}{(1+e^{-0.05\alpha})^2} - 1 \tag{2.3}$$

and as $Sig(x)|_{\Sigma_j x_j = \frac{20}{19}} = \frac{1}{2}$, $Y(x)|_{\Sigma_j x_j = \frac{20}{19}} = 1$ we have,

$$\begin{aligned}\Phi(\frac{20}{19}) &= \frac{19}{20} \cdot \frac{3}{2}(n-1) - \left(\frac{19}{20}\right)^2 \frac{\alpha}{4} - n \\ &= 0.425n - 0.225625\alpha - 1.425 \end{aligned} \tag{2.4}$$

In equation (2.3), we have that $\Phi(1) > 0$ for every $n \geq 3$ and $\alpha \geq 100$. In contrast, in equation (2.4), we have that $\Phi(\frac{20}{19}) < 0$ for every $n \geq 0$ and $\alpha \geq 2.2(n-1)$. Therefore, for every $n \geq 3$ and $\alpha \geq \max(2.2(n-1), 100)$ we have that $\Phi(1) > 0$ and $\Phi(\frac{20}{19}) < 0$. By the Value Theorem, since $\Phi(s)$ is continuous in $s \in (1, \frac{20}{19}]$, there exists $\hat{s} \in (1, \frac{20}{19})$ such that $\Phi(\hat{s}) = 0$. Note that Claim 1 and Claim 2 imply that $\Phi(s) \neq 0$ if $s \notin [1, \frac{20}{19}]$. Therefore, $\Phi(s) = 0$ if and only if $s \in (1, \frac{20}{19}]$. Since any stable state x^* implies $\frac{\partial u_i(x^*)}{\partial x_i^*} = 0$, we get that $\Phi(S(x^*)) = 0$. Thus, it must be that $S(x^*) \in (1, 20/19)$. $\square$

We now show that there is a unique stable state. This follows from the following auxiliary lemma.

Lemma 2. *Let $G(x)$ be an $n \times n$ matrix such that $G_{ij} = \frac{\partial^2 u_i(x)}{\partial x_i \partial x_j}$. If $S(x) \in [1, 20/19]$ and $a > 100$, then $G + G^T$ is negative definite.*

Proof. To prove that $G + G^T$ is strictly negative definite we show that both G and G^T are strictly negative definite. To show that G is strictly negative definite we define two matrices A and B such that $G = A + B$, and show that A is negative semidefinite and B is strictly negative definite. Using the same arguments we show that $G^T = A^T + B^T$ is strictly negative definite.

First, we express G formally

$$G_{ij} \triangleq \frac{\partial^2 u_i(x)}{\partial x_i \partial x_j} = -\frac{sig(x)+1}{S(x)^2} + \frac{2x_i(sig(x)+1)}{S(x)^3} - \frac{\alpha Y(x) sig^2(x)}{S(x)^3} + \frac{4\alpha x_i Y(x) sig^2(x)}{S(x)^4}$$
$$- \frac{x_i \alpha^2 Y(x) sig^2(x)}{S(x)^5} + \frac{2\alpha^2 x_i Y(x)^2 sig^3(x)}{S(x)^5} \tag{2.5}$$

Note that for every $j \neq i, k \neq i$, we have $G_{ij} = G_{ik}$ (all elements in each row are identical except for the diagonal element).

$$G_{ii} \triangleq \frac{\partial^2 u_i(x)}{\partial x_i^2} = -\frac{2(sig(x)+1)}{S(x)^2} + \frac{2x_i(sig(x)+1)}{S(x)^3} - \frac{2aY(x) sig^2(x)}{S(x)^3} + \frac{4\alpha x_i Y(x) sig^2(x)}{S(x)^4}$$
$$- \frac{x_i \alpha^2 Y(x) sig^2(x)}{S(x)^5} + \frac{2\alpha^2 x_i Y(x)^2 sig^3(x)}{S(x)^5} \tag{2.6}$$

Let A be an $n \times n$ matrix such that

$$A_{ii} \triangleq -\frac{sig(x)}{S(x)^2} - \frac{\alpha Y(x)}{S(x)^3} sig^2(x) - \frac{1}{S(x)^2}$$
$$A_{ij} \triangleq 0 \quad \text{for } j \neq i$$

Let B be an $n \times n$ matrix such that all elements in each row i of B are identical and equal to G_{ij} (recall that G_{ij} for every $j \neq i, k \neq i$). Note that indeed $G = A + B$ and $G^T = A^T + B^T$ (since A is diagonal $A = A^T$). We will show that B is negative semidefinite. We first show that $B_{ij} < 0$ as for every $i \neq j$. As $B_{ij} \triangleq G_{ij}$, simplifying (2.5) we get

$$B_{ij} = \frac{2x_i - S(x)}{S(x)^3} sig(x) + \frac{4\alpha x_i Y(x) - \alpha Y(x) S(x)}{S(x)^4} sig^2(x)$$
$$+ \frac{2\alpha^2 x_i Y(x)^2 - \alpha^2 x_i Y(x)(1+Y(x))}{S(x)^5} sig^3(x) - \frac{S(x) - 2x_i}{S(x)^3} \tag{2.7}$$
$$= \frac{x_i - r}{S(x)^3} sig(x) + \frac{\alpha Y(x)(3x_i - r)}{S(x)^4} sig^2(x) - \frac{\alpha^2 x_i Y(x)(1 - Y(x))}{S(x)^5} sig^3(x) + \frac{x_i - r}{S(x)^3} \tag{2.8}$$

where r denote $\sum_{j \neq i} x_j$.

There are two cases in (2.8):

1. If $x_i < \frac{S(x)}{4}$ then $B_{ij} < 0$ as all terms in (2.8) are negative.

2. Otherwise, we get

$$\begin{aligned} B_{ij} \leq & \frac{sig(x)}{S(x)^2} + \frac{\alpha Y(x)}{S(x)} sig^2(x) - \frac{\alpha^2 Y(x)(1-Y(x))}{4S(x)^4} sig^3(x) + \frac{1}{S(x)^2} \\ & < 2 + \frac{\alpha Y(x)}{S(x)} sig^2(x) - \frac{\alpha^2 Y(x)(1-Y(x))}{4S(x)^4} sig^3(x) < 0 \end{aligned}$$

where the last inequality is derived from the fact that by the specified constraints, the co-domain of $Y(x)$ is contained $[e^{-5}, 1]$, and $a \geq 100$.

Since all the elements in each row in B are identical and negative, the first eigenvalue $\lambda_1 = \sum_i \beta_i < 0$, and the other eigenvalues are zero. Therefore, B is negative semidefinite. In the same way, we can show that B^T is negative definite. In addition, A is a negative definite matrix as all its eigenvalues are negative. Note that $A = A^T$. Since G is a sum of a negative definite matrix A and negative semidefinite matrix B, G is negative definite. Similarly, since G^T is a sum of negative definite matrix A^T and negative semidefinite matrix B^T, G^T is negative definite. Hence, the matrix $G + G^T$ is negative definite as well. □

Lemma 3. *$u_i(x)$ is quasi-concave in x_i.*

Proof. To prove quasi-concavity, we show that $u_i(x)$ continuously increases in x_i up to some point $x_i = x_i^*$ and then $u_i(x)$ continuously decreases in x_i (assuming that $S(x_{-i}) \leq \frac{20}{19}$, as otherwise $u_i(x)$ decreases in x_i for all $x_i > 0$). We now divide the domain of $S(x)$ into four sub-domains $S(x) \leq 1$, $1 < S(x) < \frac{20}{19}$, $S(x) = \frac{20}{19}$, and $S(x) > \frac{20}{19}$. The high level idea is as follows: We first claim that

1. $\frac{\partial u_i(x)}{\partial x_i} > 0$ for $S(x) \leq 1$.

2. $\frac{\partial u_i(x)}{\partial x_i} < 0$ for $S(x) = \frac{20}{19}$.

By the Intermediate Value Theorem of $\frac{\partial u_i(x)}{\partial x_i}$, it must be that there is x^* for which $S(x^*) \in (1, \frac{20}{19})$ such that $\frac{\partial u_i(x^*)}{\partial x_i} = 0$. We will now show that for $1 < S(x) < \frac{20}{19}$, $\frac{\partial^2 u_i(x)}{\partial x_i^2} < 0$. This guarantees that there is a unique x_i such that $\frac{\partial u_i(x)}{\partial x_i} = 0$ (given a fixed sum of the others' sending rates). We will then show that $\frac{\partial u_i(x)}{\partial x_i} < 0$ for $S(x) > \frac{20}{19}$. Quasi-concavity will follow.
To simplify the argument for each subdomain, we substitute the terms in $\frac{\partial u_i(x)}{\partial x_i}$ with variables T_1, T_2, and T_3 as follows: $T_1(x_i, x_{-i}) = \frac{Sig(x)+1}{S(x)}$

$T_2(x_i, x_{-i}) = 1 - \frac{x_i}{S(x)}$
$T_3(x_i, x_{-i}) = \frac{ax_i e^{a(0.95 - \frac{1}{S(x)})} Sig^2(S(x))}{S(x)^3}$
By (2.1), $\frac{\partial u_i(x)}{\partial x_i}$ can be expressed as $T_1 T_2 - T_3 - 1$. We now elaborate for each subdomain:

1. $\frac{\partial u_i(x)}{\partial x_i} > 0$ for $S(x) \leq 1$ (the case of no bottleneck. Then i benefits by increasing the sending rate - see Claim 1).

2. $\frac{\partial u_i(x)}{\partial x_i} < 0$ for $S(x) = \frac{20}{19}$ (it can be verified that $T_3 > T_1 \cdot T_2$ for $a > 100$).

3. $\frac{\partial^2 u_i(x)}{\partial x_i^2} < 0$ for $S(x) \in (1, \frac{20}{19})$. This follows as:

   (a) T_1 decreases in x_i, as both $Sig(x)$ and $\frac{1}{S(x)}$ decreases in x_i, and T_2 decreases in x_i as $-\frac{x_i}{x_i+s}$ decreases in x_i for every constant $s > 0$. Therefore, $T_1 \cdot T_2$ decreases in x_i.

   (b) T_3 increases in x_i as

$$\begin{aligned}
\frac{\partial T_3(x)}{\partial x_i} &= \frac{aY(x)Sig^2(x)}{S(x)^3}\left(1 - \frac{axY(x)Sig(x)}{S(x)^2} - \frac{3x}{S(x)} + \frac{ax}{S(x)^2}\right) && (2.9)\\
&\geq \frac{aY(x)Sig^2(x)}{S(x)^3}\left(1 - \frac{ax}{4S(x)^2} - \frac{3x}{S(x)} + \frac{ax}{S(x)^2}\right) && (2.10)\\
&\geq \frac{aY(x)Sig^2(x)}{S(x)^3}\left(1 + \frac{3x}{S(x)}\left[\frac{a}{4} \cdot \frac{19}{20} - 1\right]\right) && (2.11)\\
&\geq 0 && (2.12)
\end{aligned}$$

   where (2.9) follows since $Y(x) \cdot Sig(x) \leq \frac{1}{4}$ (this holds since $f(t) \triangleq \frac{t}{(1+t)^2} \leq \frac{1}{4}$ for all $t > 0$ and hence $Y(x)Sig(x) \equiv f(Y(x)) \leq \frac{1}{4}$). Inequality (2.10) follows as $S(x) \leq \frac{20}{19}$, and inequality 2.11 is due to $a > 100, S(x) > 0, Y(x) > 0$, and $Sig(x) > 0$.

4. $\frac{\partial u_i(x)}{\partial x_i} < 0$ for every $x > \frac{20}{19}$. This is true as the sigmoid function drops exponentially fast to zero as x_i increases. That is
$\frac{\partial u_i(x)}{\partial x_i}|_{S(x) > \frac{20}{19}} \approxeq \frac{1}{S(x)}\left(1 - \frac{x_i}{S(x)}\right) - 1 < \frac{1}{S(x)} - 1 < 0.$

□

Proof of Theorem 1. Let $R_i = \{x_i \in \mathbb{R} | S(x) \in (1, \frac{20}{19}]\}$ be the possible sending rates of sender i. Since the constraints on each sender's rate are concave (that is $x_i > 0$ for all $i \in N$), $R = \prod_{i \in N} R_i$ is a convex set. Hence, as $G + G^T$ is a negative definite matrix (Lemma 2), according Rosen's theorem (Theorem 6 in [68]), the stable state in R is unique.

Theorem 2. *In the unique stable state $x^* = (x_1^*, \ldots, x_n^*)$ the sending rates of all senders are equal, i.e., $x_1^* = x_2^* = \ldots = x_n^*$.*

Proof. Suppose, for point of contradiction, that in the unique stable configuration there exist two senders, i and j, such that $x_i^* \neq x_j^*$. Observe, however, that as all senders have the same utility function, the state obtained from x^* by setting the sending rates of sender i to be x_j^* and the sending rate of sender j to be x_i^* must also be stable—a contradiction to the uniqueness of x^*. Thus, $x_1^* = x_2^* = \ldots = x_n^*$. $\square$

2.3.2 Convergence to a Stable State

Next, we show that a simple control algorithm can converge to that equilibrium. At each time step t, each sender j updates its sending rate according to $x_j^{t+1} = x_j^t(1+\varepsilon)$ if j's utility would improve by unilaterally making this change, and $x_j^{t+1} = x_j^t(1-\varepsilon)$ otherwise. Here $\varepsilon > 0$ is a small number. In this model, senders concurrently update their rates, but each sender decides based on a utility comparison as if it were the only one changing. This model does not explicitly consider measurement delay, but we believe it is a reasonable simplification. We also conjecture the model can be relaxed to allow for asynchrony. We discuss in Chapter 3 our implementation with practical optimizations of the control algorithm and show the theoretical assumptions can be relaxed in real-world.

The utility of sender i can be expressed as

$$u_i(x_i, x_{-i}) = x_i \cdot \left(\frac{Sig(x_i, x_{-i}) + 1}{S(x_i, x_{-i})} - 1 \right)$$

where x_{-i} denotes the sending rates of the other senders,
i.e., $x_{-i} \triangleq (x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$.

Claim 3. *Given a configuration of sending rates (state) $x = (x_1, \ldots, x_n)$, where $\Sigma_j x_j > 1$, and two senders i, j, $\frac{\partial u_i(x)}{\partial x_i} > \frac{\partial u_j(x)}{\partial x_j}$ if $x_j > x_i$.*

Proof. For any sender $p \in N$, $u_p(x)$ depends only on x_p and $S(x)$. Since $S(x)$ is the same for all senders, in order to see how the utility changes for for a given state, we explore how the utility increases given a fixed $S(x)$ (and denote $S(x) \triangleq S$). Therefore,

$$\frac{\partial u_p(x)}{\partial x_p} = \frac{Sig(S)+1}{S} - 1 - x_p \left(\frac{1}{S} + \frac{ae^{a(0.95-\frac{1}{S})} Sig^2(S)}{S^3} \right) \tag{2.13}$$

Since, (2.13) is a (linear) decreasing function in x_p, the claim follows. □

2.3.3 PCC dynamics

Let x_i^t denote the sending rate (sending rate) of sender i at time t. The PCC dynamic is as follows: at each time step t, j updates his sending rate according to

$$x_j^{t+1} = \begin{cases} x_j^t(1+\varepsilon) & \text{if } u_j(x_j^t(1+\varepsilon), x_{-j}) > u_j(x_j^t(1-\varepsilon), x_{-j}) \\ x_j^t(1-\varepsilon) & \text{if } u_j(x_j^t(1-\varepsilon), x_{-j}) > u_j(x_j^t(1+\varepsilon), x_{-j}) \end{cases}$$

where $\varepsilon > 0$ is small (about 0.01). We denote the increase step of x_j by $x_j \uparrow$ and the decrease step by $x_j \downarrow$.

We define an *indifferent state* $v^* = (\hat{v}, ..., \hat{v})$ to be a configuration of sending rates in which all PCC senders are indifferent between increasing or decreasing their sending rate. We will show that such a state indeed exists. Let $\bar{x}$ be the average sending rate, i.e., $\bar{x} \triangleq \frac{S(x)}{n}$.

Lemma 4. *(monotonicity) Let x be a state, and let $i, j \in n$ be a pair of senders such that $x_j > x_i$.*

1. *If $x_j \uparrow$ then $x_i \uparrow$.*

2. *If $x_i \downarrow$ then $x_j \downarrow$.*

Proof. Note that

$$\begin{aligned} u_i(x_i(1+\varepsilon), x_{-i}) - u_i(x_i, x_{-i}) = & \int_{x_i}^{x_i(1+\varepsilon)} \tfrac{\partial u_i(l, x_{-i})}{\partial l}\, dl \\ u_i(x_i(1-\varepsilon), x_{-i}) - u_i(x_i, x_{-i}) = & \int_{x_i}^{x_i(1-\varepsilon)} \tfrac{\partial u_i(l, x_{-i}))}{\partial l}\, dl \end{aligned}$$

Hence, $x_i^{t+1} = x_i^t(1+\varepsilon)$, if and only if

$$\int_{x_i(1-\varepsilon)}^{x_i(1+\varepsilon)} \frac{\partial u_i(l, x_{-i})}{\partial l}\, dl > 0 \tag{2.14}$$

Suppose that $x_j \uparrow$. We now check two complementary cases:

1. If $\frac{\partial u_i(x_i(1+\varepsilon),x_{-i})}{\partial l} > 0$, this implies that $\frac{\partial u_i(l,x_{-i})}{\partial l} > 0$ for all $l \in (x_i(1-\varepsilon), x_i(1+\varepsilon))$. Hence, condition (2.14) holds. Note that in this case the condition holds independently of j's action.

2. Otherwise (that is $\frac{\partial u_i(x_i(1+\varepsilon),x_{-i})}{\partial l} < 0$), we show that for every $x_j > x_i$, sender i benefits more by increasing than sender j, that is,

$$\int_{x_j(1-\varepsilon)}^{x_j(1+\varepsilon)} \frac{\partial u_j(l,x_{-j})}{\partial l}\, dl - \int_{x_i(1-\varepsilon)}^{x_i(1+\varepsilon)} \frac{\partial u_i(l,x_{-i})}{\partial l}\, dl < 0 \tag{2.15}$$

Specifically:

$$\int_{x_j(1-\varepsilon)}^{x_j(1+\varepsilon)} \frac{\partial u_j(l,x_{-j})}{\partial l}\, dl - \int_{x_i(1-\varepsilon)}^{x_i(1+\varepsilon)} \frac{\partial u_i(l,x_{-i})}{\partial l}\, dl \tag{2.16}$$
$$< \int_{x_i-\varepsilon x_j}^{x_i+\varepsilon x_j} \frac{\partial u_i(l,x_{-i})}{\partial l}\, dl - \int_{x_i-\varepsilon x_i}^{x_i+\varepsilon x_i} \frac{\partial u_i(l,x_{-i})}{\partial l}\, dl \tag{2.17}$$
$$= \int_{x_i+\varepsilon x_i}^{x_i+\varepsilon x_j} \frac{\partial u_i(l,x_{-i})}{\partial t}\, dl - \int_{x_i-\varepsilon x_i}^{x_i-\varepsilon x_j} \frac{\partial u_i(l,x_{-i})}{\partial l}\, dl \tag{2.18}$$
$$< 0 \tag{2.19}$$

Inequality (2.17) follows from Claim 3. Inequality 2.19 is derived as follows: The left term satisfies $\int_{x_i+\varepsilon x_i}^{x_i+\varepsilon x_j} \frac{\partial u_i(l,x_{-i})}{\partial t}\, dl < 0$. To see this, observe that $x_i(1+\varepsilon)$ is on the "decreasing part" of the function and therefore $x_j(1+\varepsilon)$ is on the "decreasing part". Hence $\frac{\partial u_i(l,x_{-i})}{\partial l} < 0$ for all $l \in (x_i+\varepsilon x_i, x_i+\varepsilon x_j)$. The right term satisfies $\int_{x_i-\varepsilon x_i}^{x_i-\varepsilon x_j} \frac{\partial u_i(l,x_{-i})}{\partial l}\, dl > 0$. This is derived from the assumption that $x_j \uparrow$, which implies that $\frac{\partial u_i(l,x_{-i})}{\partial l} > 0$ for all $l \in (x_i - \varepsilon x_j, x_i - \varepsilon x_i)$.

□

From the monotonicity property we can derive the following possible dynamics:

Corollary 1. 1. *All senders increase their sending rate (called total increase).*

2. *All senders decrease their sending rate (called total decrease)*

3. *There is a "cross-over" rate p such that each sender i with $x_i > p$ decrease its sending rate, and each sender i with $x_i < p$ increase its sending rate.*

Another important property relates to the average of the senders' sending rate.

Lemma 5. *The PCC dynamics are such that*

1. *If $\bar{x} \geq \hat{v}$ and $x_i > \hat{v}$ then $x_i \downarrow$.*

2. *If $\bar{x} \leq \hat{v}$ and $x_i < \hat{v}$ then $x_i \uparrow$.*

Proof. We only prove the first statement as the second statement relies a symmetric argument. Let x be a state such that $\bar{x} \geq \hat{v}$ and let x' be a state such that $\bar{x}' = \hat{v}$. Suppose that $x_j = x'_j = \hat{v}$ for some sender j. Since $\bar{x}' \leq \bar{x}$, the link at x is more congested than at x' and so the increase in utility from $x_j \uparrow$ is lower then the increase in utility from $x'_j \uparrow$. Since x'_j is indifferent, i.e., $u_j(x'_j \uparrow, x_{-j}) = u_j(x'_j \downarrow, x_{-j})$ (as $x_j = \hat{v}$ and $\bar{x}' = \hat{v}$), it must be that j benefits by $x_j \downarrow$. If $x_i > \hat{v}$ then $x_i > x_j$, and hence by Lemma 4, i benefits by $x_i \downarrow$. $\square$

We now classify the possible dynamics as follows:

1. *Increasing cycle.* A sequence of states $R = (x_1, .., x_T)$, where $T \geq 2$, is called increasing cycle if all senders in all states in R increase their sending rate, and at least one sender in x_{T+1} decreases its sending rate.

2. *Decreasing cycle.* A sequence of states $R = (x_1, .., x_T)$, where $T \geq 2$ (except for the special case of backing up which then $T \geq 3$), is called decreasing cycle if all senders at all state in R decreases their sending rate, and at least one sender in x_{T+1} increase its sending rate.

3. *ZigZag*: all senders increase their sending rate and then decrease their sending rate alternately. We call the series of the zigzag actions a cycle. A series of states $x^T, x^{T+1}, ..., x^{T+a}$ is called ZigZag cycle if $x^{t+1} = x^t(1+r)$ (namely all senders increase their sending rate) for $t \in (1, 3, 5, ..., a-1)$ and $x^{t+1} = x^t(1-\varepsilon)$ (namely all senders decrease their sending rate) for $t \in (2, 4, 6, ..., a)$. The ZigZag cycle is the maximal series of 'zigzags' in the sense that X^{T-1} increases its sending rate and x^{T+a+1} decreases it sending rate. A state x^t that obtained by all senders increase their sending rate is called *high peak* and a state x^t that obtained by all senders decrease their sending rate is called *low peak.*

4. *Crossover*: This is a one step dynamics. Let S_1 and S_2 be a partition of N such that for each $i \in S_1$ and $j \in S_2$, $x_i < x_j$. Under crossover we have $x_{i\in S_1} \uparrow$ and $x_{j \in S_2} \downarrow$.

5. *Backing up*: Two consecutive steps of total increase that begins when $\bar{x} \in (1-r)\hat{v}, \frac{1}{1+r}\hat{v}))$.

6. *ZigZag - Backing up - ZigZag (ZBZ)*: A ZigZag cycle that is followed by two consecutive states in which all senders increase their sending rate after a ZigZag cycle (backing up) and then another ZigZag.

Let $\hat{C} \triangleq ((1-r)\hat{v}, (1+r)\hat{v})$ be a domain of sending rates.

Lemma 6. *In a ZigZag cycle R, for all $t \in R$, $\bar{x}^t \in \hat{C}$.*

Proof. Suppose, by contradiction, that $\bar{x} \notin \hat{C}$. W.l.o.g, suppose that for t, $\bar{x}^t = a$ where $a > (1+\varepsilon)\hat{v}$. There are two cases:

1. Suppose that x^t is a high peak. Thus at $t-1$ there is total increase. Hence, $\bar{x}^{t-1} = \frac{a}{1+\varepsilon} > \hat{v}$, and w.l.o.g., the maximal sender $x^{t-1}_{max} > \bar{x}$. But then $x^{t-1}_{max} > \hat{v}$ and $\bar{x}^{t-1} > \hat{v}$. But by the property of Lemma 5 $x^{t-1}_{max} \uparrow$ contradiction.

2. Otherwise, suppose that x^t is a low pick, (i.e., total increase at time t). Again, w.l.o.g., the maximal sender $x^t_{max} > \bar{x}$, and then $x^{t+1}_{max} > \hat{v}$. Since $\bar{x}^t > \hat{v}$ and $x^{t+1}_{max} > \hat{v}$ by the property of Lemma 5 $x^{t+1}_{max} \downarrow$ - contradiction.

□

The backing up dynamics can occur after a ZigZag cycle (as the sending rates of all senders decrease after each pair of total increase-total decrease cycle) until $\bar{x} \in (1-\varepsilon)\hat{v}, \frac{1}{1+\varepsilon}\hat{v})$ which then ZigZag cannot occur (otherwise it contradicts Lemma 6). If after a backing up there is another ZigZag cycle then it is ZBZ. Endless ZBZ (namely, zigzag cycle that ends in backing up and then return to another zigzag cycle and so on infinitely) is an unwanted behavior (as it contradicts convergence). But we claim that this 'bad' dynamics cannot occur in general.

Lemma 7. *There is no endless ZBZ (for almost all possible values of r).*

Proof. We claim this for a general r (however it might happen for an exact value of ε). Each zigzag cycle start in a slightly different $\bar{x}$ after backing up. Hence at some zigzag cycle, one of the high peaks of $\bar{x}$ in the ZigZag cycle will be sufficiently close to $\hat{v}$ and so there will be a crossover (by lemma 9) (this, for instant, can be enforced by choosing irrational value for r and then in the ZBZ cycle, $\bar{x}$ forms a dense set in $\hat{C}$). □

Lemma 8. *For any initial state there is at most increasing or decreasing cycle (but not both) which begins at the initial state.*

Proof. Assume there is a decreasing cycle that begins at time t, after the following dynamics:

1. **Increasing cycle**. Then at $t-1$ there is a total increase and at $t+1$ there is a total decrease. Therefore, for every sender i, $x_i^{t+1} = (1+\varepsilon)(1-\varepsilon)x_i^{t-1} = (1-\varepsilon^2)x_i^{t-1}$, and then the average satisfies $\bar{x}^{t+1} = (1-\varepsilon^2)\bar{x}^{t-1}$ as well. Because $x_i^{t+1} < x_i^{t-1}$ and $\bar{x}^{t+1} < \bar{x}^{t-1}$, as each sender $x_i^{t-1} \uparrow$, x_i^{t+1} would benefit more than x_i^{t-1} by increasing sending rate. Thus, as $x_i^{t-1} \uparrow$ it must be that $x_i^{t+1} \uparrow$ - contradiction to assumption.

2. **ZigZag** (or ZBZ). Followed by the exact same argument as the increasing cycle case, it is impossible that a decreasing cycle will occur after ZigZag.

3. **Crossover**. Let x_{min} be the sender with minimal sending rate at time $t-1$. Crossover at time $t-1$ implies that $x_{min}^{t-1} \uparrow$. Therefore, $x_{min}^{t+1} = (1+\varepsilon)(1-\varepsilon)x_{min}^{t-1} = (1-\varepsilon^2)x_{min}^{t-1}$. By crossover, there are senders at time $t-1$ that decrease sending rate and there are sender that increase sending rate (in particular x_{min}), hence using similar argument as in the increasing cycle case, we get $\bar{x}^{t+1} < (1-\varepsilon^2)\bar{x}^{t-1}$ (but now its a strict inequality instead of equality) and $x_{min}^{t+1} \uparrow$ - contradiction to assumption.

The proof that an increasing cycle cannot occur after Decreasing cycle, ZigZag, and crossover is followed by the symmetric argument (unless a condition for backing up is met and then comparing x^{t-1} and x^{t+2}) □

We call the increasing or decreasing cycle a *starting phase*.

Therefore, by Lemma 8 after the initial phase of possible Increasing or Decreasing cycle, the only possible dynamics are ZigZag, Crossover and Backing-up (less probable).

Crossover

The following lemma tells us when crossover occurs.

Lemma 9. *If $\bar{x}$ is sufficiently close to $\hat{v}$ (including $\hat{v}$ itself) then crossover occurs.*

Proof. Let p be a crossover pivot such that if $x_j > p$ then $x_j \downarrow$ and if $x_j < p$ then $x_j \uparrow$. Because the utility function is quasi concave and $(\hat{v}, ..., \hat{v})$ is a stable state, if $\bar{x} = \hat{v}$ then each sender with sending rate lower than $\hat{v}$ will benefit by increasing the sending rate, and each sender with sending rate grater than $\hat{v}$ will benefit by decreasing the sending rate. Hence, $p = \hat{v}$. In addition, note that increasing $\bar{x}$ continuously result in a continuous decrease in p (as the link becomes more congested it become less attractive to increase the sending rate for all senders). Since any continuous change in $\bar{x}$ cause a continuous change in p, for infinitesimal change in $\hat{x}$, $p > 0$. However, note that if $\bar{x}$ is "far" from $\hat{v}$ then $p < 0$ and there is no crossover. □

The following claim (which follows immediately by the definition of crossover) says that if the two sending rate are not too close then after crossover they become closer. Formally,

Claim 4. *Let t be a time step in which a crossover occurs, where $x_i^t \downarrow$ and $x_j^t \uparrow$. If $x_i^t(1-\varepsilon) > x_j^t(1+\varepsilon)$, then the distance between the two senders decreases (i.e., $|x_i^{t+1} - x_j^{t+1}| < |x_i^t - x_j^t|$).*

We now check what if the distance of two sending rate by crossover can increase when x_i is "close" to x_j.

Claim 5. *If two sender i, j such that $x_i^t > x_j^t$, undergo crossover s.t. $x_i^t \downarrow$ and $x_j^t \uparrow$, and $x_i^t < x_j^t(1+\varepsilon)^2$ then $x_j^{t+1} < x_i^{t+1}(1+\varepsilon)^2$.*

Proof. Since the sender are close, i.e., ($x_i^t(1-\varepsilon) < x_j^t(1+\varepsilon)$) their distance will increase when i and j switch their order" (i.e., $x_j^{t+1} > x^{t+1}$). The distance between them is maximized at time $t+1$ when x_i^t is infinitesimally close to x_j^t. Then, after the crossover we get $x_j^{t+1} = x_j^{t+1}\frac{1+\varepsilon}{1-\varepsilon}$. As ε is small, by Taylor approximation $1+\varepsilon \simeq \frac{1}{1-\varepsilon}$, and thus if $x_i^{t+1} < xt + 1_j(1+\varepsilon)^2$ the two sender switch orders and then $x_j^{t+1} < x_i^{t+1}(1+\varepsilon)^2$. □

Let x_{max} be the maximal sending rate, and x_{min} be the maximal sending rate.

Lemma 10. *There is a finite number of crossovers such that after that $x^{t+1}_{max} < x^{t+1}_{min}(1+\varepsilon)^2$, and $x^{t+1}_{min} > x^{t+1}_{max}(1-\varepsilon)^2$.*

Proof. By Claim 4, all sender that are ”far”, their distance is reduced, and any pair of senders that are close (i.e., for $x_j < x_i$ $x^{t+1}_i < x^{t+1}_j(1+\varepsilon)^2$) remain close. Since ZigZag does not increase the distance between senders, after finite number of steps we get $x^{t+1}_{max} < x^{t+1}_{min}(1+\varepsilon)^2$. Getting $x^{t+1}_{min} > x^{t+1}_{min}(1-\varepsilon)^2$ we use the symmetric argument. □

If state x satisfies $x^{t+1}_{max} < x^{t+1}_{min}(1+\varepsilon)^2$, and $x^{t+1}_{min} > x^{t+1}_{min}(1-\varepsilon)^2$ then we call this convergent state.

Lemma 11. *If all senders are at convergent state, then $\bar{x} \in \hat{C}$*

Proof. Since by Lemma 6, during ZigZag we always have $\bar{x} \in \hat{C}$, and ZBZ is possible only when $\bar{x} \in ((1-\varepsilon)\hat{v}, \frac{1}{1+\varepsilon}\hat{v})$ and hence after backing up $\bar{x} < (1+\varepsilon)\hat{v} \in \hat{C}$, we only need to show that after a crossover it holds that $\bar{x} \in \hat{C}$. Suppose w.l.o.g. that after a crossover $\bar{x}^{t+1} > \hat{C}$. This implies that $\bar{x}^t > \hat{v}$, as otherwise it wouldn't increase to this high sending rate at time $t+1$ (we assume $\bar{x}^t \in \hat{C}$ as, w.l.o.g, it occurs after a ZigZag cycle). Since the average increases at time t, there must be a sender i that increases it sending rate, and by lemma 5, it must be that $x_i < \hat{v}$. Since all senders are at convergent state it implies that $x^{t+1}_i > \bar{x}$ (as all senders that are above $\hat{v}$ decrease their sending rate). Therefore, i increases it rate by more than $(1+\varepsilon)$ in one step (as $x^t_i < \hat{v}$ and $x^t_i > \hat{v}(1+\varepsilon)$), which is clearly impossible. □

Theorem 3. *There is a time t' such that for all $t > t'$, $x^t_i \in ((1-\varepsilon)^2\hat{v}, (1+\varepsilon)^2\hat{v})$ for each $i \in N$.*

Proof. By Lemma 8, after the initial phase the only possible dynamics are ZigZag (or ZBZ) and crossover. By Lemma 7 there is no endless ZBZ. This implies that there is infinite number of crossovers. Therefore, by Lemma 10 after a finite number of steps all senders are in convergent state. Since by Lemma 11 $\bar{x} \in \hat{C}$, the theorem follows □

It might seem surprising that PCC uses *multiplicative* rate increase and decrease, yet achieves convergence and fairness. If TCP used MIMD, in an idealized network senders would often get the same back-off signal at the same time, and so would take the *same multiplicative decisions in lockstep*, with the ratio of their rates never changing. In PCC, senders make *different* decisions.

Consider a 100 Mbps link with sender A at rate 90 Mbps and B at 10 Mbps. When A experiments with slightly higher and lower rates $(1 \pm \varepsilon)90$ Mbps, it will find that it should decrease its rate to get higher utility because when it sends at higher than equilibrium rate, the loss rate dominates the utility function. However, when B experiments with $(1 \pm \varepsilon)10$ Mbps it finds that loss rate increase is negligible compared with its improved throughput. This occurs precisely because B is responsible for little of the congestion. In fact, this reasoning (and the formal proof of the game dynamics) is *independent of the step size* that the flows use in their experiments: PCC senders move towards the convergence point, even if they use a heterogeneous mix of AIMD, AIAD, MIMD, MIAD or other step functions. Convergence behavior does depend on the choice of utility function, however.

2.4 Utility Function: Source of Flexibility

PCC carries a level of flexibility beyond TCP's architecture: the same learning control algorithm can cater to different applications' heterogeneous objectives (e.g. latency vs. throughput) by using different utility functions. For example, under TCP's architecture, latency based protocols [56, 75] usually contain different hardwired mapping algorithms than loss-based protocols [43]. Therefore, without changing the control algorithm, as Sivaraman et al. [69] recently observed, TCP has to rely on different in-network active queue management (AQM) mechanisms to cater to different applications' objectives because *even with fair queueing*, TCP is blind to applications' objectives. However, by literally changing one line of code that describes the utility function, PCC can flip from "loss-based" (§2.3) to "latency-based" (§3.4.4) and thus caters to different applications' objectives without the complexity and cost of programmable AQMs [69]. There are even more powerful potentials of PCC's pluggable utility function and we will discuss that in much more details in Chapter 4.

2.5 Related work

It has long been clear that TCP lacks enough information, or the right information, to make optimal rate control decisions. XCP [52] and RCP [26] solved this by using explicit feedback from the network to directly set the sender's rate. But this requires new protocols, router hardware, and packet header formats, so deployment is rare.

Numerous designs modify TCP, e.g. [27, 43, 56, 75, 79], but fail to acheive consistent high performance, because they still inherit TCP's hardwired mapping architecture. As we evaluate in § 3.4, they partially mitigate TCP's problems in the specially assumed network scenarios but still suffer from performance degradation when their assumptions are violated. As another example, FAST TCP [75] uses prolonged latency as a congestion signal for high BDP connections. However, it models the network queue in an ideal way and its performance degrades under RTT variance [25], incorrect estimation of baseline RTT [71] and when competing with loss-based TCP protocols.

The method of Remy and TAO [70, 77] pushes TCP's architecture to the extreme: it searches through a large number of *hardwired mappings* under a network model with assumed parameters, e.g. number of senders, link speed, etc., and finds the best protocol under that simulated scenario. However, like all TCP variants, when the real network deviates from Remy's input assumption, performance degrades [70].

Works such as PCP [23] and Packet Pair Flow Control [54] utilize techniques like packet-trains [48] to probe available bandwidth in the network. However, bandwidth probing protocols do not observe real performance like PCC does and make unreliable assumptions about the network. For example, real networks can easily violate the assumptions about packet inter-arrival latency embedded in BP (e.g. latency jitter due to middleboxes, software routers or virtualization layers), rendering incorrect estimates that harm performance.

Several past works also explicitly quantify utility. Analysis of congestion control as a global optimization [53, 60] implemented by a distributed protocol is not under the same framework as our analysis, which defines a utility function and finds the global Nash equilibrium. Other work explicitly defines a utility function for a congestion control protocol, either local [22] or global [70, 77]. However, the resulting control algorithms are still TCP-like hardwired mappings, whereas each PCC sender optimizes utility using a learning algorithm that obtains direct experimental evidence of how sending rate affects utility. Take Remy and TAO again as an example: there is a global optimization goal, used to guide the choice of protocol; but at the end of day the senders use hardwired control to attempt to optimize for that goal, which can fail when those assumptions are violated and moreover, one has to change the hardwired mapping if the goal changes.

Decongestion Control [67] sends at full line rate, masking loss with erasure coding. PCC is selfish, but optimizes a utility function and converges to an efficient equilibrium.

Finally, none of the aforementioned work allows the possibility of expressing different sending objectives by plugging in different utility functions as PCC does.

2.6 Conclusion

In this chapter, we present PCC's overall congestion control architecture that controls sending rate based on empirically observed performance metrics. We analytically prove that competing PCC senders, even though selfish in nature, can converge to a stable convergence point given certain kind of utility function and rate control algorithm. However, many questions remain, such as: Is it possible to transfer the theoretical insight to build a working system that actually delivers the consistent high performance promises? How to handle noisy and unreliable measurement in the network? How fast and stable the convergence process actually is? How useful is the capability of pluggable utility function? The list goes on. We give answers to these questions with the first instantiation of PCC's architecture: PCC Allegro, in the next chapter.

Chapter 3

PCC Allegro: A Strong Case for the PCC Architecture

3.1 Introduction

In this chapter, we describe the first instantiation of the PCC architecture: PCC Allegro. Transforming the above theoretical insights to real system requires solving a multitude of challenges. To realize the capability of associating a particular sending rate to the empirical performance utility value, we implement a time-window based performance monitoring framework. The performance monitoring cycle also drives the rate control process. On top of the performance monitoring framework, we implement a rate control algorithm based on the simplified algorithm used in theoretical analysis in chapter 2. In that implementation, we identify and address important practical challenges such as reactivity to changing network conditions, speed and stability tradeoff in convergence state and the reliability of the measurement statics.

We implemented PCC Allegro in userspace by adapting the UDP-based TCP skeleton in the UDT [19] package. With handling real-world complexity and therefore achieving consistent high performance as the most important goal, we experimentally evaluate PCC Allegro in both controlled and reproducible chanllenging network conditions and large-scale and real-world networks. Without tweaking its control algorithm, PCC Allegro achieves consistent high performance and significantly beats *specially engineered* TCPs in various network environments: (a.) in the wild on the global commercial Internet (often more than 10$\times$ the throughput of TCP CUBIC); (b.) inter-data center networks (5.23$\times$ vs. TCP Illinois); (c.) emulated satellite Internet links (17$\times$ vs TCP Hybla); (d.) unreliable lossy links ($10-37\times$ vs Illinois); (e.) unequal RTT of competing senders (an architectural cure to RTT unfairness); (f.) shallow buffered bottleneck links (up to 45$\times$ higher performance, or 13$\times$ less buffer to reach 90% throughput); (g.) rapidly changing networks (14$\times$ vs CUBIC, 5.6$\times$ vs Illinois). PCC Allegro performs similar to ICTCP [79] in (h.) the incast scenario

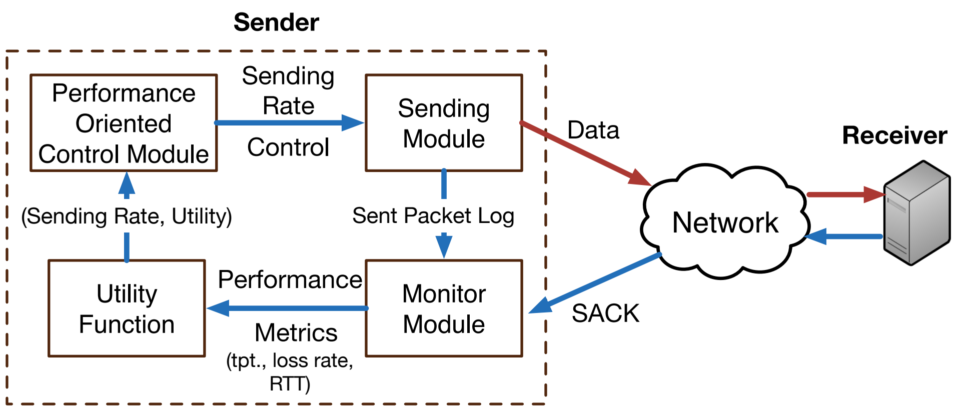


Figure 3.1: PCC Allegro prototype design

in data centers. Moreover, we studied PCC Allegro Allegro's convergence behavior to show that it achieves a better reactivity and stability tradeoff than a large number of TCP variants. We also demonstrate a case to show the potential of pluggable utility function, by using a latency sensitive utility function in the presence of in-network resource isolation. Though it is a substantial shift in architecture, PCC Allegro can be deployed by only replacing the sender-side rate control of TCP. It can also deliver real data today with a user-space implementation at `speedier.net/pcc`.

3.2 PCC Allegro System Design

Fig. 3.1 depicts PCC Allegro's modularized design. The high-level flow of events is as follows: The Sending Module sends packets at a certain rate to the receiver; the receiver sends acknowledgements, much as in TCP; the Monitor observes this feedback, informs the Sending Module of what packets should be retransmitted, and periodically combines the observed performance metrics (throughput, RTT, etc.) into values to pass to the Utility Function. The Utility Function module combines these performance metrics into a single numerical utility value, and passes a (rate, utility) pair to the Performance-oriented Control Module, which runs a learning algorithm over the stream of pairs to dynamically adjust the sending rate to maximize the utility value. We next discuss each module in more detail.

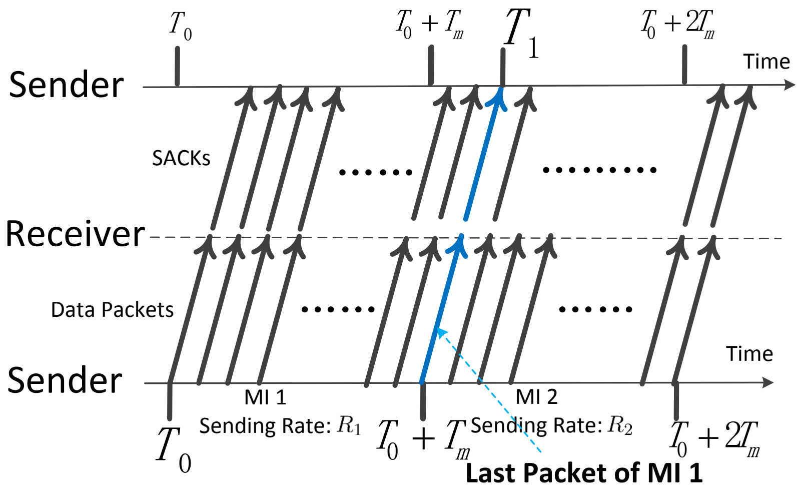


Figure 3.2: Performance monitoring process

3.2.1 Sending Module

The Sending Module's task is to send data packets at a certain rate. There is no notion of congestion window in PCC Allegro's design. The Sending Module picks data from a limited-size data buffer, which is filled by the application for new data and by the Monitor for data to be retransmitted. It does not know the type of this data (new or retransmission) and continues sending until the flow finishes. In our current UDP implementation, every packet in the queue has a Data Sequence Number, which tracks bytes in the application's data stream as in TCP. In addition, each sent packet includes a Transmission Sequence Number (TSN) that uniquely identifies the packet, so the Monitor Module can differentiate packets with the same DSN and thus collect better measurements. However, we use TSN only for convenience of prototyping and for a full kernel implementation, we believe only TCP's SACK mechanism will be enough.

3.2.2 Performance Monitoring

The Monitor Module collects statistics regarding the network's performance based on feedback messages from the receiver and informs the Sending Module about necessary retransmissions. The Monitor uses two streams of SACK coming back from the receiver for different purposes. First, the TSN's SACK is used to collect performance data, because TSN feedback tells the ground truth about what happened to each individual sent packet. With this feedback, the Monitor periodically summarizes performance into a handful of metrics including RTT, throughput, and loss rate and feeds them to the utility function module.

As shown in Fig. 3.2.2, the timeline is sliced into chunks of duration of T_m called the *Monitor Interval* (MI). When the Sending Module sends packets (new or retransmission) at a certain sending rate instructed by the Performance-oriented Control Module, the Monitor Module will remember what packets are sent out during each MI. As the SACK comes back from receiver, the Monitor will know what happened (received? lost? RTT?) to each packet sent out during an MI. Taking the example of Fig. 3.2.2, the Monitor knows what packets were sent during MI1, spanning T_0 to T_0+T_m. At time T_1, approximately one RTT after $T_0 + T_m$, it has received the SACKs for all packets sent out in MI1. The Monitor aggregates these individual SACKs into meaningful performance metrics including throughput, loss rate and average RTT. The performance metrics are then combined by a utility function; unless otherwise stated, we use the utility function of §2.3. The result of this is that we associate a control action of each MI (sending rate) with its performance result (utility). This pair forms a "micro-experiment" and is used by the performance oriented control module.

To ensure there are enough packets in one monitor interval, we set T_m to the maximum of (a) the time to send 10 data packets and (b) a uniform-random time in the range $[1.7, 2.2]$ RTT. Again, we want to highlight that PCC Allegro *does not* pause sending packets to wait for performance results, and it *does not* decide on a rate and send for a long time; packet transfer and measurement-control cycles occur continuously.

Note that the measurement results of one MI can arrive after the next MI has begun, and the control module can decide to change sending rate after processing this result. As an optimization, PCC Allegro will immediately change the rate and "re-align" the current MI's starting time with the time of rate change without waiting for the next MI.

3.2.3 Control Algorithm

We designed a practical control algorithm with the gist of the simple control algorithm described in §2.3.

Starting State

PCC Allegro starts at rate $2 \cdot MSS/RTT$ (i.e., $3KB/RTT$) and doubles its rate at each consecutive monitor interval (MI), like TCP. Unlike TCP, PCC Allegro does not exit this starting phase because of a packet loss. Instead, it monitors the utility result of each rate doubling action. Only when the utility decreases, PCC Allegro exits the starting state, returns to the previous rate which had higher utility (i.e., half of the rate), and enters the *Decision Making State*. PCC Allegro could use other more aggressive startup strategies, but such improvements could be applied to TCP as well. Note that, in starting phase, PCC Allegro does not wait for utility result comes back and doubles the rate, but doubles the rate in every interval and record all utility measurement to fall back to the right rate.

Decision Making State

Decisions with noisy measurements. PCC Allegro's experiments on the live network will tend to move its rate in the direction that improves utility. But it may also make some incorrect decisions. In the example above, if the loss is random non-congestion loss, it may randomly occur that loss is substantially higher when PCC Allegro tests rate 105 Mbps, causing it to pick the lower rate. Alternately, if the loss is primarily due to congestion from this sender, unpredictable external events (perhaps another sender arriving with a large initial rate while PCC Allegro is testing rate 100 Mbps) might cause a particular 105 Mbps microexperiment to have higher throughput and lower loss rate. More generally, the network might be changing over time for reasons unrelated to the sender's action. This adds noise to the decision process: PCC Allegro will on average move in the right direction, but may make some unlucky errors.

We improve PCC Allegro's decisions with **multiple randomized controlled trials (RCTs)**. Rather than running two tests (one each at 100 and 105 Mbps), we conduct four in randomized order—e.g. perhaps $(100, 105, 105, 100)$. PCC Allegro only picks a particular rate as the winner if utility is higher in *both* trials with that rate. This produces increased confidence in a causal connection between PCC Allegro's action and the observed utility. If results are inconclusive, so each rate "wins" in one test, then PCC Allegro maintains its current rate, and we may have reached a local optimum (details follow later).

As we will see, without RCTs, PCC Allegro already offers a dramatic improvement in performance and stability compared with TCP, but RCTs further reduce rate variance by up to 35%. Although it might seem that RCTs will double convergence time, this is not the case because they help PCC Allegro make *better* decisions; overall, RCTs improve the stability/convergence-speed tradeoff space.

With RCT, PCC Allegro takes four consecutive MIs and divides them into two pairs (2 MIs each). For each pair, PCC Allegro attempts a slightly higher rate $r(1+\varepsilon)$ and slightly lower rate $r(1-\varepsilon)$, each for one MI, in random order. After the four consecutive trials, PCC Allegro changes the rate back to r and keeps aggregating SACKs until the Monitor generates the utility value for these four trials. For each pair $i \in 1,2$, PCC Allegro gets two utility measurements U_i^+, U_i^- corresponding to $r(1+\varepsilon), r(1-\varepsilon)$ respectively. If the higher rate consistently has higher utility ($U_i^+ > U_i^- \ \forall i \in \{1,2\}$), then PCC Allegro adjusts its sending rate to $r_{new} = r(1+\varepsilon)$; and if the lower rate consistently has higher utility then PCC Allegro picks $r_{new} = r(1-\varepsilon)$. However, if the results are inconclusive, e.g. $U_1^+ > U_1^-$ but $U_2^+ < U_2^-$, PCC Allegro stays at its current rate r and re-enters the Decision Making State with larger experiment granularity, $\varepsilon = \varepsilon + \varepsilon_{min}$. The granularity starts from ε_{min} when it enters the Decision Making State for the first time and will increase up to ε_{max} if the process continues to be inconclusive. This increase of granularity helps PCC Allegro avoid getting stuck due to noise. Unless otherwise stated, we use $\varepsilon_{min} = 0.01$ and $\varepsilon_{max} = 0.05$.

Rate Adjusting State

Rate Adjusting State: Assume the new rate after Decision Making is r_0 and $dir = \pm 1$ is the decided moving direction. In each monitor period, PCC Allegro adjusts its rate in that direction faster and faster, setting the new rate r_n as:

$$r_n = r_{n-1} \cdot (1 + n \cdot \varepsilon_{min} \cdot dir) \tag{3.1}$$

However, if $U(r_n) < U(r_{n-1})$, PCC Allegro decreases its rate to r_{n-1} and enters the *Decision Making State.*

3.3 Deployment Viability Analysis

Despite being a significant architectural shift, PCC Allegro needs only isolated changes. **No router support:** unlike ECN, XCP [52], and RCP [33], there are no new packet fields to be standardized and processed by routers. **No new protocol:** The packet format and semantics can simply remain as in TCP (SACK, hand-shaking and etc.). **No receiver change:** TCP SACK is enough feedback. What PCC Allegro does change is the control algorithm within the sender.

The remaining concern is how PCC Allegro safely replaces and interacts with TCP. We observe that there are many scenarios where critical applications suffer severely from TCP's poor performance and PCC Allegro can be safely deployed by fully replacing or being isolated from TCP. First, **when a network resource is owned by a single entity** or can be reserved for it, the owner can replace TCP entirely with PCC Allegro. For example, some Content Delivery Network (CDN) providers use dedicated network infrastructure to move large amounts of data across continents [11, 12], and scientific institutes can reserve bandwidth for exchanging huge scientific data globally [35]. Second, PCC Allegro can be used in challenging network conditions **where per-user or per-tenant resource isolation is enforced** by the network. Satellite Internet providers are known to use per-user bandwidth isolation to allocate the valuable bandwidth resource [18]. For data centers with per-tenant resource isolation [42, 64, 65], an individual tenant can use PCC Allegro safely within its virtual network to address problems such as incast and improve data transfer performance between data centers.

The above applications, where PCC Allegro can fully replace or be isolated from TCP, are a significant opportunity for PCC Allegro. But in fact, **PCC Allegro does not depend on any kind of resource isolation to work.** In the public Internet, the key issue is TCP friendliness. Using PCC Allegro with the utility function described in §2.3 is not TCP friendly. However, we also study the following utility function which incorporates latency: $u_i(x) = (T_i \cdot Sigmoid_\alpha(L_i - 0.05) \cdot Sigmoid_\beta(\frac{RTT_{n-1}}{RTT_n} - 1) - x_i \cdot L_i)/RTT_n$ where RTT_{n-1} and RTT_n are the average RTT of the previous and current MI, respectively. In §3.4.3 we show that with this utility function, PCC Allegro successfully achieves TCP friendliness in various network conditions. Indeed, it is even possible for PCC Allegro to be TCP friendly while achieving much higher performance in challenging scenarios (by taking advantage of the capacity TCP's poor control algorithm leaves unused). Overall, this is a promising direction but we only take the first steps in this paper.

It is still possible that individual users will, due to its significantly improved performance, decide to deploy PCC Allegro in the public Internet with the default utility function. It turns out that the default utility function's unfriendliness to TCP is comparable to the common practice of opening parallel TCP connections used by web browsers today [4], so it is unlikely to make the ecosystem *dramatically* worse for TCP; see §3.4.3.

3.4 Evaluation

We demonstrate PCC Allegro's architectural advantages over the TCP family through diversified, large-scale and real-world experiments: §3.4.1: PCC Allegro achieves its design goal of **consistent high performance**. §3.4.2: PCC Allegro can actually achieve much **better fairness and convergence/stability tradeoff** than TCP. §3.4.3: PCC Allegro is **practically deployable** in terms of flow completion time for short flows and TCP friendliness. §3.4.4: PCC Allegro has a huge potential to flexibly **optimize for applications' heterogenous objectives** with fair queuing in the network rather than more complicated AQMs [69].

3.4.1 Consistent High Performance

We evaluate PCC Allegro's performance under 8 real-world challenging network scenarios with with phno algorithm tweaking for different scenarios. Unless otherwise stated, all experiments using the same default utility function of §2.3. In the first 7 scenarios, PCC Allegro significantly outperforms specially engineered TCP variants.

Inter-Data Center Environment

Here we evaluate PCC Allegro's performance in scenarios like inter-data center data transfer [7] and dedicated CDN backbones [11] where **network resources can be isolated or reserved for a single entity**.

The GENI testbed [8], which has reservable bare-metal servers across the U.S. and reservable bandwidth [10] over the Internet2 backbone, provides us a representative evaluation environment. We choose 9 pairs of GENI sites and reserved **800Mbps** end-to-end dedicated bandwidth between

Transmission Pair	RTT (ms)	PCC Allegro (Mbps)	SABUL (Mbps)	CUBIC (Mbps)	Illinois (Mbps)
GPO → NYSERNet	12	818	563	129	326
GPO → Missouri	47	624	531	80.7	90.1
GPO → Illinois	35	766	664	84.5	102
NYSERNet → Missouri	47	816	662	108	109
Wisconsin → Illinois	9	801	700	547	562
GPO → Wisc.	38	783	487	79.3	120
NYSERNet → Wisc.	38	791	673	134	134
Missouri → Wisc.	21	807	698	259	262
NYSERNet → Illinois	36	808	674	141	141

Table 3.1: PCC Allegro significantly outperforms TCP in inter-data center environments.

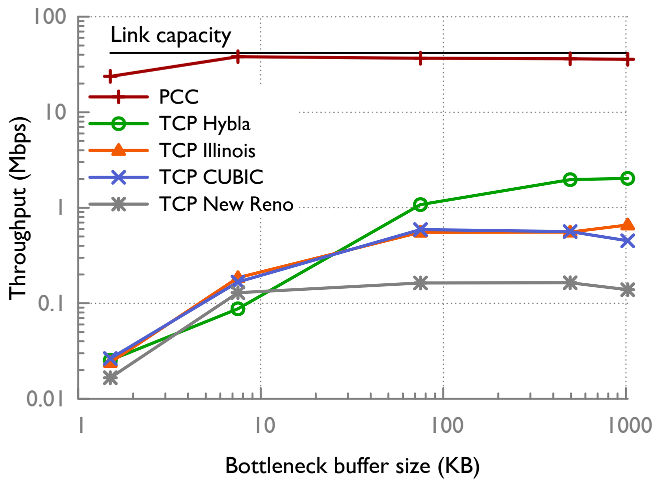


Figure 3.3: PCC Allegro outperforms special TCP modifications on emulated satellite links

each pair. We compare PCC Allegro, SABUL [41], TCP CUBIC [43] and TCP Illinois [56] over 100-second runs.

As shown in Table 3.1, PCC Allegro significantly outperforms TCP Illinois, by 5.2× on average and up to 7.5×. It is surprising that even in this very clean network, specially optimized TCPs still perform far from optimal. We believe some part of the gain is because the bandwidth-reserving rate limiter has a small buffer and TCP will overflow it, unnecessarily decreasing rate and also introducing latency jitter that confuses TCP Illinois. (TCP pacing will not resolve this problem; §3.4.1.) On the other hand, PCC Allegro continuously tracks the optimal sending rate by continuously measuring performance.

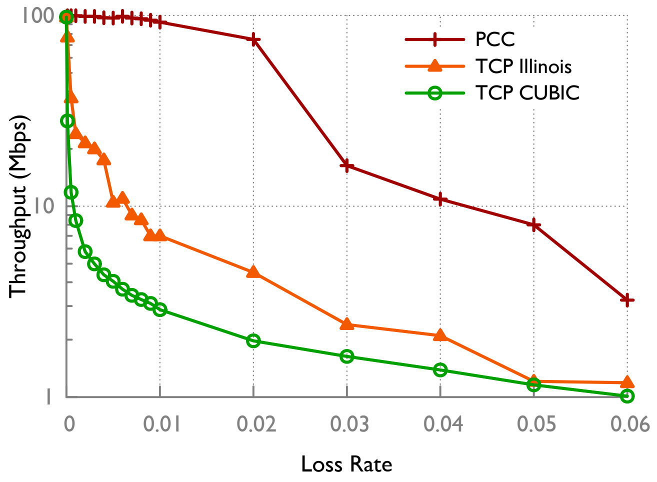


Figure 3.4: PCC Allegro is highly resilient to random loss

Satellite Links

Satellite Internet is widely used for critical missions such as emergency and military communication and Internet access for rural areas. Because TCP suffers from severely degraded performance on satellite links that have excessive latency (600ms to 1500ms RTT [17]) and relatively high random loss rate [63], special modifications of TCP (Hybla [27], Illinois) were proposed and special infrastructure has even been built [46, 72].

We test PCC Allegro against TCP Hybla (widely used in real-world satellite communication), Illinois and CUBIC under emulated satellite links on Emulab parameterized with the real-world measurements of the WINDs satellite Internet system [63]. The satellite link has **800ms RTT, 42Mbps capacity and 0.74% random loss**. As shown in Fig. 3.4.1, we vary the bottleneck buffer from 1.5KB to 1MB and compare PCC Allegro's average throughput against different TCP variants with 100 second trials. PCC Allegro achieves 90% of optimal throughput even with only a 7.5KB buffer (5 packets) at the bottleneck. However, even with 1MB buffer, the widely used TCP Hybla can only achieve 2.03Mbps which is 17× worse than PCC Allegro. TCP Illinois, which is designed for high random loss tolerance, performs 54× worse than PCC Allegro with 1MB buffer.

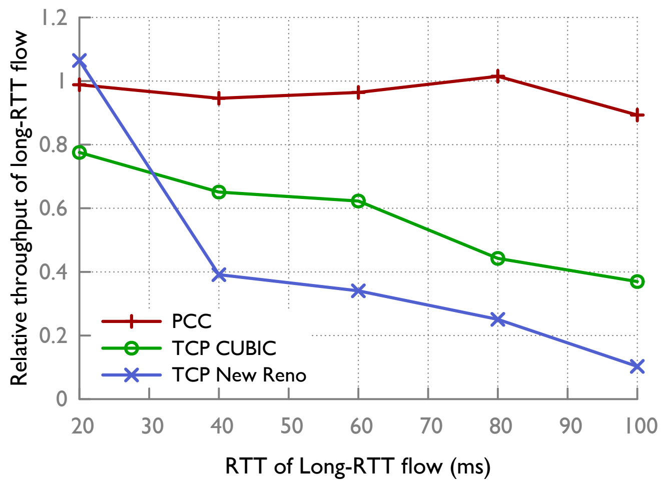


Figure 3.5: PCC Allegro achieves better RTT fairness than specially engineered TCPs

Unreliable Lossy Links

To further quantify the effect of random loss, we set up a link on Emulab with **100Mbps bandwidth, 30ms RTT and varying loss rate**. As shown in Fig. 3.4.1, PCC Allegro can reach $> 95\%$ of achievable throughput capacity until loss rate reaches 1% and shows relatively graceful performance degradation from 95% to 74% of capacity as loss rate increases to 2%. However, TCP's performance collapses very quickly: CUBIC's performance collapses to $10\times$ smaller than PCC Allegro with only 0.1% loss rate and $37\times$ smaller than PCC Allegro with 2% random loss. TCP Illinois shows better resilience than CUBIC but throughput still degrades severely to less than 10% of PCC Allegro's throughput with only 0.7% loss rate and $16\times$ smaller with 2% random loss. Again, PCC Allegro can endure random loss because it looks at real utility: unless link capacity is reached, a higher rate will always result in similar loss rate and higher throughput, which translates to higher utility.

PCC Allegro's performance does decrease to 3% of the optimal achievable throughput when loss rate increases to 6% because we are using the "safe" utility function of §2.3 that caps the loss rate to 5%[1].

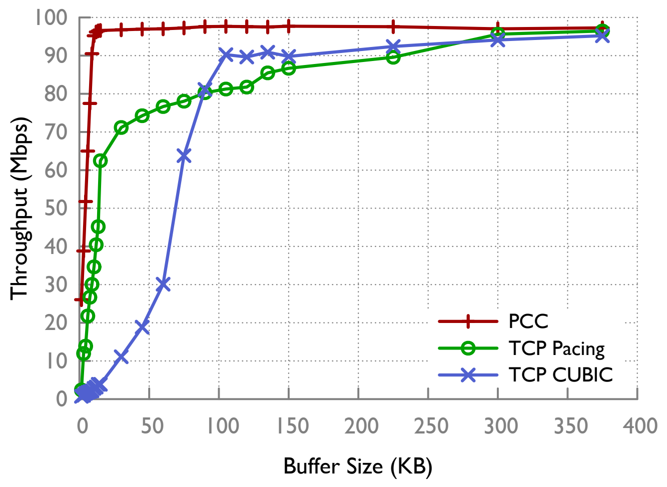


Figure 3.6: PCC Allegro efficiently utilizes shallow-buffered networks

Mitigating RTT Unfairness

For unmodified TCP, short-RTT flows dominate long-RTT flows on throughput. Subsequent modifications of TCP such as CUBIC or Hybla try to mitigate this problem by making the expansion of the congestion window independent of RTT. However, the modifications cause new problems like parameter tuning (Hybla) and severely affect stability on high RTT links (CUBIC) [43]. Because PCC Allegro's convergence is based on real performance not the control cycle length, it acts as an architectural cure for the RTT unfairness problem. To demonstrate that, on Emulab we set **one short-RTT (10ms) and one long-RTT (varying from 20ms to 100ms) network path sharing the same bottleneck link of 100Mbit/s bandwidth** and buffer equal to the BDP of the short-RTT flow. We run the long-RTT flow first for 5s, letting it grab the bandwidth, and then let the short-RTT flow join to compete with the long-RTT flow for 500s and calculate the ratio of the two flows' throughput. As shown in Fig. 3.4.1, PCC Allegro achieves much better RTT fairness than New Reno and even CUBIC cannot perform as well as PCC Allegro.

Small Buffers on the Bottleneck Link

TCP cannot distinguish between loss due to congestion and loss simply due to buffer overflow. In face of high BDP links, a shallow-buffered router will keep chopping TCP's window in half and the recovery process is very slow. On the other hand, the practice of over-buffering networks, in the

[1]Throughput does not decrease to 0% because the sigmoid function is not a clean cut-off.

fear that an under-buffered router will drop packets or leave the network severely under-utilized, results in bufferbloat [39], increasing latency. This conflict makes choosing the right buffer size for routers a challenging multi-dimensional optimization problem [38, 66, 73] for network operators to balance between throughput, latency, cost of buffer memory, degree of multiplexing, etc.

Choosing the right buffer size would be much less difficult if the transport protocol could efficiently utilize a network with very shallow buffers. Therefore, we test how PCC Allegro performs with a tiny buffer and compare with TCP CUBIC, which is known to mitigate this problem. Moreover, to address the concern that the performance gain of PCC Allegro is merely due to PCC Allegro's use of packet pacing, we also test an implementation of TCP New Reno with pacing rate of $(congestionwindow)/(RTT)$. We set up on Emulab a network path with **30ms RTT, 100Mbps bottleneck bandwidth and vary the network buffer size from 1.5KB (one packet) to 375KB ($1 \times BDP$)** and compare the protocols' average throughput over 100s.

As shown in 3.4.1, PCC Allegro only requires $6 \cdot MSS$ (9 KB) buffer to reach 90% capacity. With the same buffer, CUBIC can only reach 2% capacity and even TCP with packet pacing can only reach 30%. CUBIC requires 13× more buffer than PCC Allegro to reach 90% throughput and takes 36× more buffer to close the 10% gap. Even with pacing, TCP still requires 25× more buffer than PCC Allegro to reach 90% throughput. It is also interesting to notice that with just a phsingle-packet buffer, PCC Allegro's throughput can reach 25% of capacity, 35× higher throughput than TCP CUBIC. The reason is that PCC Allegro constantly monitors the real achieved performance and steadily tracks its rate at the bottleneck rate without swinging up and down like TCP. That means with PCC Allegro, network operators can use shallow buffered routers to **get low latency without harming throughput.**

Rapidly Changing Networks

The above scenarios are static environments. Next, we study a network that changes rapidly during the test. We set up on Emulab a network path where **available bandwidth, loss rate and RTT are all changing every 5 seconds.** Each parameter is chosen independently from a uniform random distribution with bandwidth ranging from 10Mbps to 100Mbps, latency from 10ms to 100ms and loss rate from 0% to 1%.

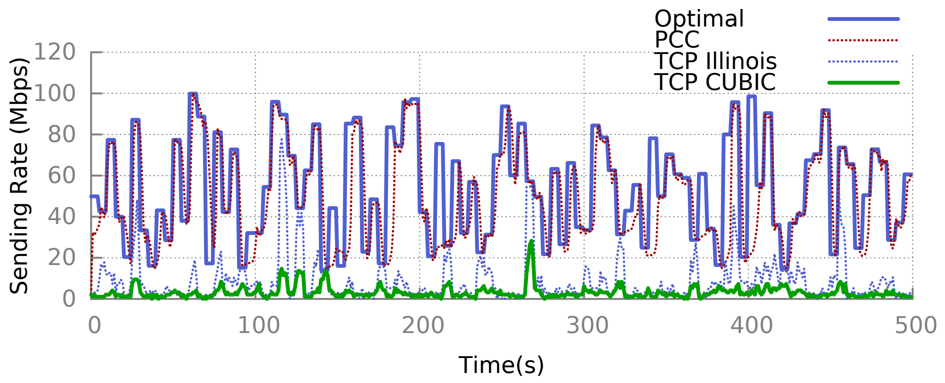


Figure 3.7: PCC Allegro can always track optimal sending rate even with drastically changing network conditions

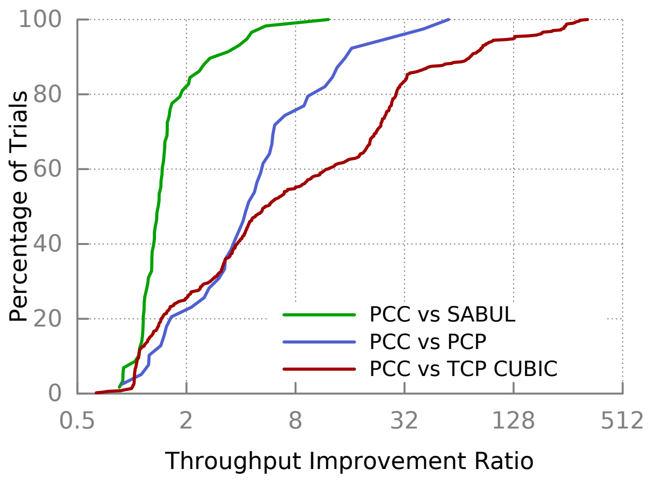


Figure 3.8: Across the public Internet, PCC Allegro has $\geq 10\times$ the performance of TCP CUBIC on 41% of tested pairs

Figure 3.7 shows available bandwidth (optimal sending rate), and the sending rate of PCC Allegro, CUBIC and Illinois. Note that we show the PCC Allegro control algorithm's chosen sending rate (not its throughput) to get insight into how PCC Allegro handles network dynamics. Even with all network parameters rapidly changing, PCC Allegro tracks the available bandwidth very well, unlike the TCPs. Over the course of the experiment (500s), PCC Allegro's throughput averages 44.9Mbps, achieving 83% of the optimal, while TCP CUBIC and TCP Illinois are $14\times$ and $5.6\times$ worse than PCC Allegro respectively.

Big Data Transfer in the Wild

Due to its complexity, the commercial Internet is an attractive place to test whether PCC Allegro can achieve consistently high performance. We deploy PCC Allegro's receiver on 85 globally distributed PlanetLab [20] nodes and senders on 6 locations: five GENI [8] sites and our local server, and ran experiments over a 2-week period in December 2013. These 510 sending-receiving pairs render a very diverse testing environment with BDP from 14.3 KB to 18 MB.

We first test PCC Allegro against TCP CUBIC, the Linux kernel default since 2.6.19; and also SABUL [19], a special modification of TCP for high BDP links. For each sender-receiver pair, we run TCP iperf between them for 100 seconds, wait for 500 seconds and then run PCC Allegro for 100 seconds to compare their average throughput. PCC Allegro improves throughput by 5.52× at the median (Fig. 3.4.1). On 41% of sender-receiver pairs, PCC Allegro's improvement is more than 10×. This is a conservative result because 4 GENI sites have 100Mbps bandwidth limits on their Internet uplinks.

We also tested two other non-TCP transport protocols on smaller scale experiments: the public releases of PCP [14, 23] (43 sending receiving pairs) and SABUL (85 sending receiving pairs). PCP uses packet-trains [48] to probe available bandwidth. However, as discussed more in §2.5, this bandwidth probing is different from PCC Allegro's control based on empirically observed action-utility pairs, and contains unreliable assumptions that can yield very inaccurate sample results. SABUL, widely used for scientific data transfer, packs a full set of boosting techniques: packet pacing, latency monitoring, random loss tolerance, etc. However, SABUL still mechanically hardwires control action to packet-level events. Fig. 3.4.1 shows PCC Allegro outperforms PCP[2] by 4.58× at the median and 15.03× at the 90th percentile, and outperforms SABUL by 1.41× at the median and 3.39× at the 90th percentile. SABUL shows an unstable control loop: it aggressively overshoots the network and then deeply falls back. On the other hand, PCC Allegro stably tracks the optimal rate. As a result, SABUL suffers from 11% loss on average compared with PCC Allegro's 3% loss.

Is PCC Allegro's performance gain merely due to TCP unfriendliness of the default

[2] $initial - rate = 1Mbps$, $poll - interval = 100\mu s$. PCP in many cases abnormally slows down (e.g. 1 packet per 100ms). We have not determined whether this is an implementation bug in PCP or a more fundamental deficiency. To be conservative, we excluded all such results from the comparison.

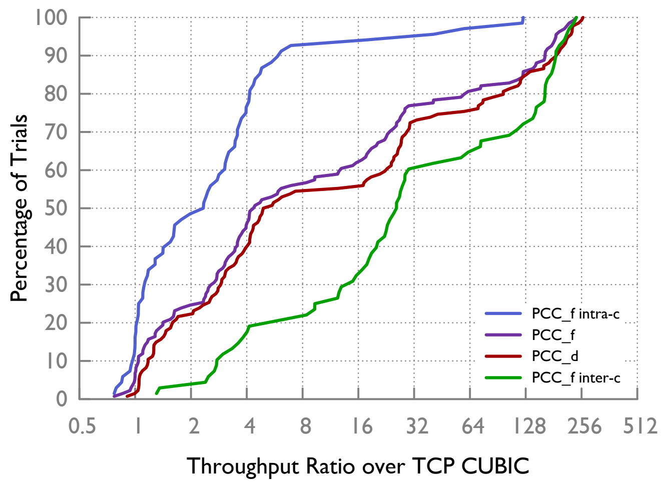


Figure 3.9: PCC Allegro's performance gain is not merely due to TCP unfriendliness

utility function? In fact, PCC Allegro's high performance gain should not be surprising given our results in previous experiments, none of which involved PCC Allegro and TCP sharing bandwidth. Nevertheless, we ran another experiment, this time with PCC Allegro using the more TCP-friendly utility function described in §3.3 with $\beta = 10$ (its TCP friendliness is evaluated in § 3.4.3), from a server at UIUC[3] to 134 PlanetLab nodes in February 2015. Fig. 3.4.1 compares the results with the default utility function (PCC_d) and the friendlier utility function (PCC_f). PCC_f still shows a median of 4.38× gain over TCP while PCC_d shows 5.19×. For 50 pairs, PCC_d yields smaller than 3% higher throughput than PCC_f and for the remaining 84 pairs, the median inflation is only 14%. The use of the PCC_f utility function does not fully rule out the possibility of TCP unfriendliness, because our evaluation of its TCP friendliness (§3.4.3) does not cover all possible network scenarios involved in this experiment. However, it is highly suggestive that the performance gain is not merely due to TCP unfriendliness.

Instead, the results indicate that PCC Allegro's advantage comes from its ability to deal with complex network conditions. In particular, geolocation revealed that the large-gain results often involved cross-continent links. On cross-continent links (68 pairs), PCC_f yielded a median gain of 25× compared with 2.33× on intra-continent links (69 pairs). We believe TCP's problem with cross-continent links is not an end-host parameter tuning problem (e.g. sending/receiving buffer size), because there are paths with similar RTT where TCP can still achieve high throughput with

[3]The OS was Fedora 21 with kernel version 3.17.4-301.

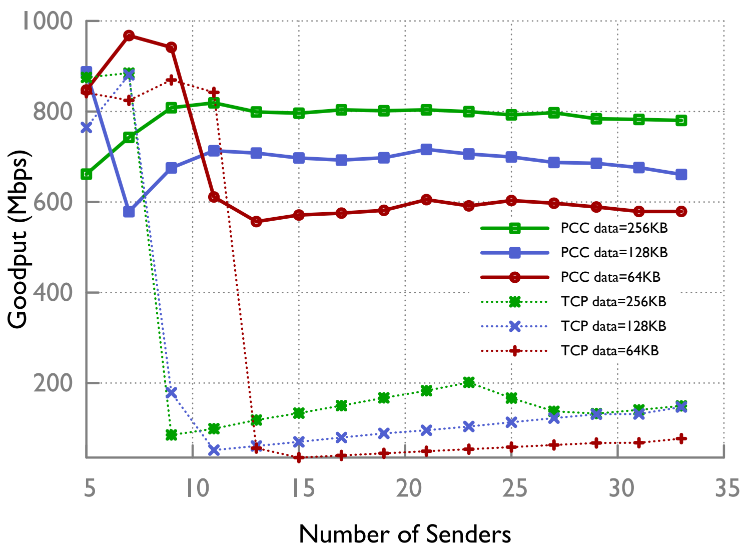


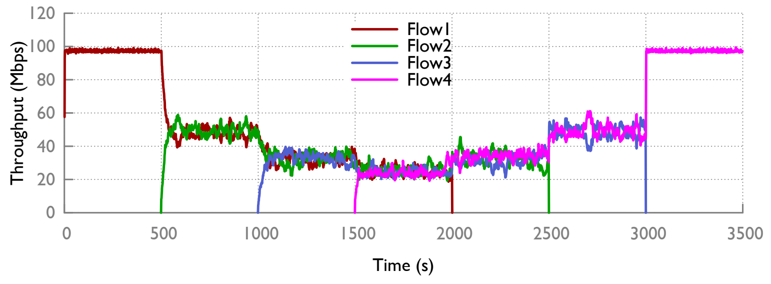
Figure 3.10: PCC Allegro largely mitigates TCP incast in a data center environment

identical OS and configuration.

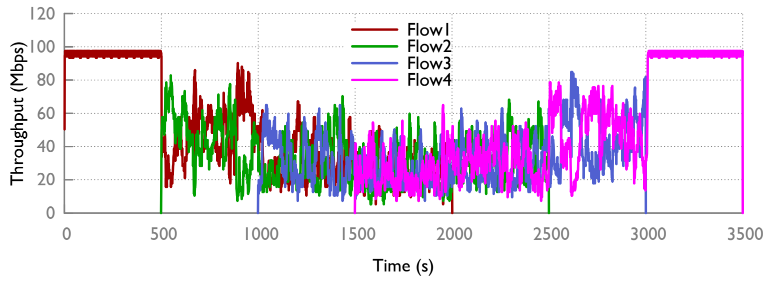
Incast

Moving from wide-area networks to the data center, we now investigate TCP incast [29], which occurs in high bandwidth and low latency networks when multiple senders send data to one receiver concurrently, causing throughput collapse. To solve the TCP incast problem, many protocols have been proposed, including ICTCP [79] and DCTCP [21]. Here, we demonstrate PCC Allegro can achieve high performance under incast without special-purpose algorithms. We deployed PCC Allegro on Emulab [76] with **33 senders and 1 receiver**.

Fig. 3.4.1 shows the goodput of PCC Allegro and TCP across various flow sizes and numbers of senders. Each point is the average of 15 trials. When incast congestion begins to happen with roughly $\geq$ 10 senders, PCC Allegro achieves roughly 60-80% of the maximum possible goodput, or 7-8$\times$ that of TCP. Note that ICTCP [79] also achieved roughly 60-80% goodput in a similar environment. Also, DCTCP's goodput degraded with increasing number of senders [21], while PCC Allegro's is very stable.



(a) PCC Allegro maintains a stable rate with competing senders



(b) TCP CUBIC shows high rate variance and unfairness at short time scales

Figure 3.11: PCC Allegro's dynamics are much more stable than TCP CUBIC with senders competing on a FIFO queue

3.4.2 Dynamic Behavior of Competing Flows

We proved in §2.3 that with our "safe" utility function, competing PCC Allegro flows converge to a fair equilibrium from any initial state. In this section, we experimentally show that **PCC Allegro is much more stable, more fair and achieves a better tradeoff between stability and reactiveness than TCP.** PCC Allegro's stability can immediately translate to benefits for applications such as video streaming where stable rate in presence of congestion is desired [50].

PCC Allegro is More Fair and Stable Than TCP

The data transmission of the four pairs initiates sequentially with a 500s interval and each pair transmits continuously for 2000s. Fig. 3.11 shows the rate convergence process for PCC Allegro and CUBIC respectively with 1s granularity. It is visually obvious that PCC Allegro flows converge much more stably than TCP, which has surprisingly high rate variance. Quantitatively, we compare PCC Allegro's and TCP's fairness ratio (Jain's index) at different time scales (Fig. 3.4.2). Selfishly

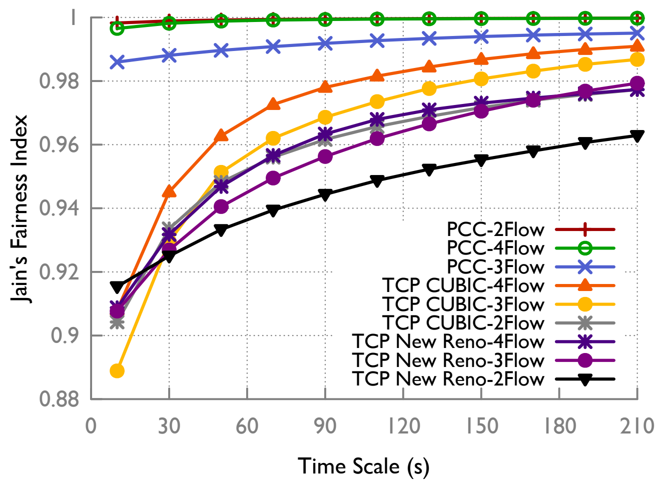


Figure 3.12: PCC Allegro achieves better fairness and convergence than TCP CUBIC

competing PCC Allegro flows achieve better fairness than TCP at all time scales.

PCC Allegro has better Stability-Reactiveness tradeoff than TCP

Intuitively, PCC Allegro's control cycle is "longer" than TCP due to performance monitoring. Is PCC Allegro's significantly better stability and fairness achieved by severely sacrificing convergence time?

We set up two sending-receiving pairs sharing a bottleneck link of 100Mbps and 30ms RTT. We conduct the experiment by letting the first flow, flow A, come in the network for 20s and then let the second flow, flow B, begin. We define the convergence time in a "forward-looking" way: we say flow B's phconvergence time is the smallest t for which throughput in each second from t to $t + 5s$ is within ±25% of the ideal equal rate. We measure stability by measuring the standard deviation of throughput of flow B for $60s$ after convergence time. All results are averaged over 15 trials. PCC Allegro can achieve various points in the stability-reactiveness trade-off space by adjusting its parameters: higher step size ε_{min} and lower monitor interval T_m result in faster convergence but higher throughput variance. In Fig. 3.13, we plot a trade-off curve for PCC Allegro by choosing a range of different settings of these parameters.[4] There is a clear convergence speed and stability trade-off: higher ε_{min} and lower T_m result in faster convergence and higher variance and vice versa.

[4]We first fix ε_{min} at 0.01 and vary the length of T_m from 4.8×RTT down to 1×RTT. Then we fix T_m at 1×RTT and vary ε_{min} from 0.01 to 0.05. This is not a full exploration of the parameter space, so other settings might actually achieve better trade-off points.

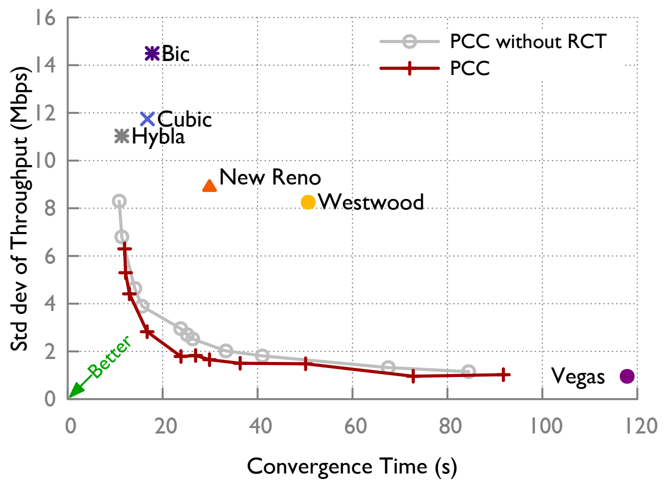


Figure 3.13: PCC Allegro has better reactiveness-stability tradeoff than TCP, particularly with its RCT mechanism

We also show six TCP variants as individual points in the trade-off space. The TCPs either have very long convergence time or high variance. On the other hand, PCC Allegro achieves a much better trade-off. For example, PCC Allegro with $T_m = 1.0 \cdot RTT$ and $\varepsilon_{min} = 0.02$ achieves the same convergence time and 4.2× smaller rate variance than CUBIC.

Fig. 3.13 also shows the **benefit of the RCT mechanism** described in §3.2.3. While the improvement might look small, it actually helps most in the "sweet spot" where convergence time and rate variance are both small, and where improvements are most difficult and most valuable. Intuitively, with a long monitor interval, PCC Allegro gains enough information to make a low-noise decision even in a single interval. But when it tries to make reasonably quick decisions, multiple RCTs help separate signal from noise. Though RCT doubles the time to make a decision in PCC Allegro's Decision State, the convergence time of PCC Allegro using RCT only shows slight increase because it makes phbetter decisions. With $T_m = 1.0 \cdot RTT$ and $\varepsilon_{min} = 0.01$, RCT trades 3% increase in convergence time for **35% reduction in rate variance**.

3.4.3 PCC Allegro is Deployable

A Promising Solution to TCP Friendliness

To illustrate that PCC Allegro does not have to be TCP unfriendly, we evaluate the utility function proposed in § 3.3. We initiate two competing flows on Emulab: a phreference flow running TCP

		30ms	60ms	90ms
$\beta = 10$	10Mbit/s	0.94	0.75	0.67
	50Mbit/s	0.74	0.73	0.81
	90Mbit/s	0.89	0.91	1.01
$\beta = 100$	10Mbit/s	0.71	0.58	0.63
	50Mbit/s	0.56	0.58	0.54
	90Mbit/s	0.63	0.62	0.88

Table 3.2: PCC Allegro can be TCP friendly

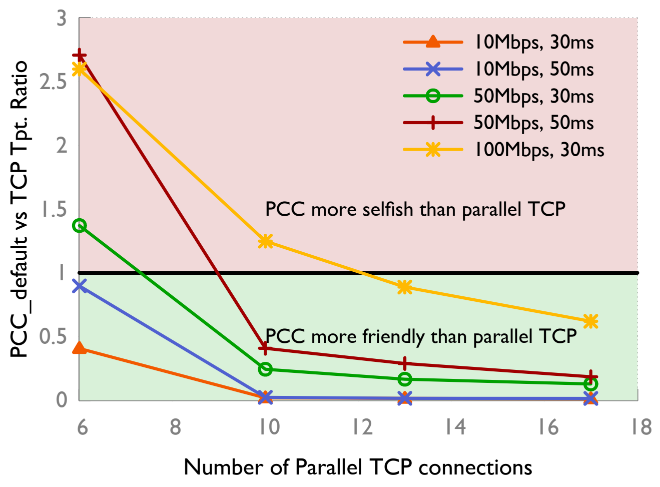


Figure 3.14: The default PCC Allegro utility function's TCP unfriendliness is similar to common selfish practice

CUBIC, and a phcompeting flow running either TCP CUBIC or PCC Allegro, under various bandwidth and latency combinations with bottleneck buffer always equal to the BDP. We compute the ratio of average (over five runs) of throughput of reference flow when it competes with CUBIC, divided by the same value when it competes with PCC Allegro. If the ratio is smaller than 1, PCC Allegro is more friendly than CUBIC. As shown in Table 3.2, PCC Allegro is already TCP friendly and with $\beta = 100$, PCC Allegro's performance is dominated by TCP. We consider this only a first step towards a TCP friendliness evaluation because these results also indicate PCC Allegro's friendliness can depend on the network environment. However, this initial result shows promise in finding a utility function that is sufficiently TCP friendly while also offering higher performance (note that this same utility function achieved higher performance than TCP in § 3.4.1).

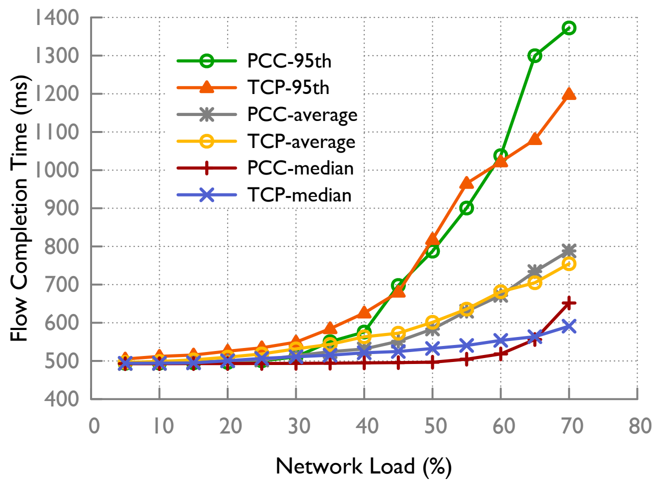


Figure 3.15: PCC Allegro can achieve flow completion time for short flows similar to TCP

TCP Friendliness of Default Utility Function

Applications today often adopt "selfish" practices to improve performance [4]; for example, Chrome opens between 6 (default) and 10 (maximum) parallel connections and Internet Explorer 11 opens between 13 and 17. We compare the unfriendliness of PCC Allegro's default utility function with these selfish practices by running two competing streams: one with a single PCC Allegro flow and the other with parallel TCP connections like the aforementioned web browsers. Fig. 3.4.3 shows the ratio of PCC Allegro's throughput to the total of the parallel TCP connections, over 100 seconds averaging over 5 runs under different bandwidth and RTT combinations. As shown in Fig. 3.4.3, PCC Allegro is similarly aggressive to 13 parallel TCP connections (IE11 default) and more friendly than 17 (IE11 maximum). Therefore, even using PCC Allegro's default utility function in the wild may not make the ecosystem dramatically worse for TCP. Moreover, simply using parallel connections cannot achieve consistently high performance and stability like PCC Allegro and initiating parallel connections involves added overhead in many applications.

Flow Completion Time for Short Flows

Will the "learning" nature of PCC Allegro harm flow completion time (FCT)? In this section, we resolve this concern by showing that with a startup phase similar to TCP, PCC Allegro achieves similar FCT for short flows.

We set up a link on Emulab with 15 Mbps bandwidth and 60ms RTT. The sender sends short flows of 100KB each to receiver. The interval between two short flows is exponentially distributed with mean interval chosen to control the utilization of the link. As shown in Fig. 3.4.3, with network load ranging from 5% to 75%, PCC Allegro achieves similar FCT at the median and 95th percentile. The 95th percentile FCT with 75% utilization is 20% longer than TCP. However, we believe this is a solvable engineering issue. The purpose of this experiment is to show PCC Allegro does not fundamentally harm short flow performance. There is clearly room for improvement in the startup algorithm of all these protocols, but optimization for fast startup is intentionally outside the scope of this paper because it is a largely separate problem.

3.4.4 Flexiblity of PCC Allegro: An Example

In this section, we show a unique feature of PCC Allegro: expressing different data transfer objectives by using different utility functions. Because TCP is blind to the application's objective, a deep buffer (bufferbloat) is good for throughput-hungry applications but will build up long latency that kills performance of interactive applications. AQMs like CoDel [62] limits the queue to maintain low latency but degrades throughput. To cater to different applications' objective with TCP running on end hosts, it has been argued that programmable AQMs are needed in the network [69]. However, PCC Allegro can accomplish this with simple per-flow fair queuing (FQ). We only evaluate this feature in a per-flow fair queueing (FQ) environment; with a FIFO queue, the utility function may (or may not) affect dynamics and we leave that to future work. Borrowing the evaluation scenario from [69], an interactive flow is defined as a long-running flow that has the objective of maximizing its throughput-delay ratio, called the phpower. To make our point, we show that PCC + Bufferbloat + FQ has the same power for interactive flows as PCC + CoDel + FQ, and both have higher power than TCP + CoDel + FQ.

We set up a transmission pair on Emulab with 40Mbps bandwidth and 20ms RTT link running a CoDel implementation [6] with AQM parameters set to their default values. With TCP CUBIC and two simultaneous interactive flows, TCP + CoDel + FQ achieves $493.8 Mbit/s^2$, which is $10.5\times$ more power than TCP + Bufferbloat + FQ ($46.8 Mbit/s^2$).

For PCC Allegro, we use the following utility function modified from the default to express the objective of interactive flows: $u_i(x_i) = (T_i \cdot Sigmoid(L_i - 0.05) \cdot \frac{RTT_{n-1}}{RTT_n} - x_i L_i)/RTT_n$ where RTT_{n-1} and RTT_n are the average RTT of the previous and current MIs, respectively. This utility function expresses the objective of low latency and avoiding latency increase. With this utility function, we put PCC Allegro into the same test setting of TCP. Surprisingly, PCC + Bufferbloat + FQ and PCC + CoDel + FQ achieve essentially the same power for interactive flows ($772.8Mbit/s^2$ and $766.3Mbit/s^2$ respectively). This is because PCC Allegro was able to keep buffers very small: we observed phno packet drop during the experiments even with PCC + CoDel + FQ so PCC's self-inflicted latency never exceeded the latency threshold of CoDel. That is to say, CoDel becomes useless when PCC Allegro is used in end-hosts. Moreover, PCC + Bufferbloat + FQ achieves 55% higher power than TCP + CoDel + FQ, indicating that even with AQM, TCP is still suboptimal at realizing the applications' transmission objective.

3.5 Conclusion

This chapter made the case that Performance-oriented Congestion Control, in which senders control their rate based on direct experimental evidence of the connection between their actions and performance, offers a promising new architecture to achieve consistent high performance. Within this architecture, many questions remain. One major area is in the choice of utility function: Is there a utility function that provably converges to a Nash equilibrium while being TCP friendly? Does a utility function which incorporates latency—clearly a generally-desirable objective—provably converge and experimentally perform as well as the default utility function used in most of our evaluation? More generally, alternate utility functions are a largely unexplored field. Also, the rate control algorithm, though with many important practical challenges addressed, still has a big room to improve, especially regarding its reactivity to changing networks and also convergence speed and stability tradeoff. We approach these important unresolved issues in Chapter 4.

Chapter 4

PCC Vivace: Online-Learning Congestion Control

4.1 Introduction

Along with the proposal of PCC and PCC Allegro, there has been a surge of interest in both academia and industry in innovating congestion control on the Internet [28, 40, 51, 55, 58, 59, 77, 78, 80]. This trend has made it more apparent than ever that today's prevalent congestion control scheme, the TCP family, is besieged by numerous significant challenges.

Summarizing from our experiences after the initial proposal of PCC architecture, we consider modern congestion control protocols need to meet three fundamental requirements. First and foremost, a congestion control architecture should be able to efficiently utilize network resources under varying and complex network conditions. Meeting this requirement involves satisfying many important properties, e.g., avoiding network congestion, attaining low latency, achieving consistently high performance in the presence of "non-congestion loss" [28], under very long RTT cross-continent links, in drastically dynamic networks such as WiFi and LTE links, etc. Second, congestion control should guarantee quick convergence to a stable and fair rate-configuration when multiple senders compete over network resources. This desideratum is particularly important for applications like high quality or virtual reality video streaming. Last, a congestion control scheme should be easy and safe (e.g., sufficiently friendly to existing protocols) to deploy in practice.

As articulated and evaluated in Chapter 2 and Chapter 3, traditional TCP congestion control [24, 43, 56] fails to satisfy the first two requirements.

To overcome the inherent deficiencies of traditional TCP congestion control architecture, along with PCC Allegro, recent proposals, including Remy [77] and BBR [28], investigate new approaches to this challenge. Remy is carefully designed to salvage TCP's congestion control scheme by replacing the human designer with an (offline) machine learning scheme that identifies the best hardwired mapping for the certain network conditions (e.g., bandwidths and RTT ranges). While attaining significant improvement in performance, Remy-generated TCPs are inherently prone to degraded performance when the actual network conditions deviate from input assumptions [70].

PCC Allegro and BBR, in contrast to Remy, adapt transmission rates in response to aggregated performance metrics such as throughput, latency, and loss rate [28, 32]. Yet, despite the vast improvement in performance over TCP, both protocols fall short of meeting the requirements of an ideal protocol. BBR takes a white-box network-modeling approach, translating change patterns in performance measurements (e.g., increase in delivery rate) to presumed network conditions, and still reacting to these presumed network conditions in a hardwired manner. Our experimental results show that this approach is plagued by the same predicament as TCP: when the model of the network does not reflect reality, performance can suffer severely. PCC Allegro, unlike BBR, builds on our proposed PCC architecture. However, as argued in § 4.2, PCC Allegro is a suboptimal realization of the general PCC online learning architecture and yields sub-optimal performance in many important environments. PCC Allegro and BBR also both fail to achieve optimal low latency and exhibit far-from-ideal tradeoffs between convergence speed and stability. Specifically, BBR exhibits high rate variance and high packet loss rate upon convergence, whereas PCC Allegro's convergence time is long. Lastly, PCC Allegro is highly aggressive towards TCP, while BBR, though designed with TCP-friendliness in mind, is even more aggressive towards TCP when the network conditions deviate from its design assumptions. Though valid arguments of evolution paths are made in § 3.3, TCP friendliness still inevitably poses a barrier to fast and wide deployments of these new protocols.

To address the above limitations, we draw inspiration from literature on online optimization [37, 44, 81]. We present PCC Vivace, a novel congestion control scheme. PCC Vivace replaces the two essential architectural components of PCC: the rate-control algorithm and the utility function framework. First, PCC Vivace employs provably optimal gradient-ascent-based online optimization to achieve quick utilization of spare network capacity, swift reaction to network changes, and fast and stable convergence. Second, PCC Vivace relies on a new, learning-theory-informed framework for utility derivation that incorporates crucial considerations such as latency minimization and TCP friendliness. Our utility framework also exposes a more broadly applicable fundamental tradeoff between random loss tolerance and packet loss at convergence.

Turning the above theoretical insights into operational reality involves overcoming significant engineering challenges, including: (1) transforming online optimization algorithms from machine learning literature into practical rate-control schemes by striking a delicate balance between reactivity and stability; and (2) gathering useful statistics about performance via online measurements in the face of very limited measurement time, changing network conditions, noisy network feedback, and beyond.

Extensive experimentation with PCC Vivace, BBR, PCC Allegro, and TCP in controlled environments, real residential Internet scenarios, and with video-streaming applications, demonstrates that PCC Vivace significantly improves upon all other evaluated congestion control schemes. First, PCC Vivace achieves significantly improved performance. Some highlights are: a. in long-RTT satellite networks, 40% higher throughput than BBR; b. in rapidly changing network conditions, 70% less packet loss and 72.5% higher throughput than PCC Allegro and around 20% median throughput gain on over BBR; c. under LTE-like network conditions, 97% lower latency inflation than PCC Allegro and 87.2% lower than BBR; d. 91% less queue length needed to maintain low latency and almost lossless, compared to BBR; e. on real-world home Internet connections, at least 11% higher performance than BBR; f. 90% less video buffering time than BBR. Second, in terms of convergence, compared to PCC Allegro, PCC Vivace provides an important new provable convergence guarantee while being latency-aware. It also converges about 2× faster than PCC Allegro and is 2× more stable than BBR. Finally, PCC Vivace is significantly more friendly to TCP CUBIC than both PCC Allegro and BBR.

Beyond the merits above, PCC Vivace's expressive utility function framework can be tuned differently at different competing flows to produce predictable converged throughput ratios for each flow. This could enable centralized network control (SDN or OpenTCP [40]) to allocate network capacity intelligently, suggesting a promising direction for future research.

4.2 Rate-Control Through the Online Learning Lens

When approached from an online learning perspective, the challenges outlined in § 4.1 fall naturally into the category of online optimization in machine learning and game theory [44, 81] (a.k.a. "no-

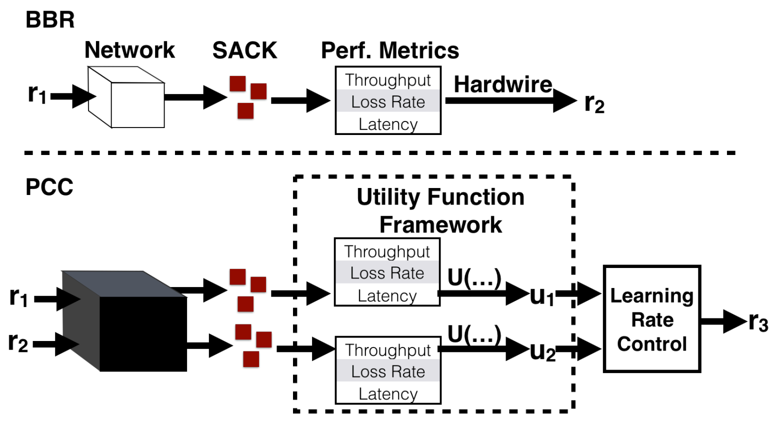


Figure 4.1: BBR and PCC's architecture difference

regret learning"). This area of research studies environments in which actors (traffic senders) repeatedly select actions in the presence of uncertainty about both each other's choices and the changing environment (e.g., the Internet) so as to optimize their self-interests (performance). The congestion control context is all the more challenging since often only very limited feedback from the network is available to the sender. Broadly, effective online learning schemes have two traits: (1) explicit quantification of the actor's goal, with a utility function; (2) learning a correlation between actions and resulting utility.

TCP congestion control was originally designed with simple control-theoretic principles, not for robust online learning. But the reality of the Internet is that TCP has to operate in a variety of complex environments. Viewed from an online learning perspective, we can see why TCP's hardwired mapping from packet-level events to rate-changes is far from optimal in practice. (1) It lacks robust quantification of a performance goal. TCP has no explicit utility function; a loose implication of utility is embedded in its hardwired mapping (situations that trigger a rate increase are considered "good"), but this is generally based on individual packet-level events, so it is vulnerable to inaccuracy especially under noisy conditions. (2) It fails to learn an association between its actions and their effect on the performance goal – for example, whether the loss rate is indeed decreased when reducing the sending rate (which it may not be if, for instance, loss is the result of PHY-layer corruption or a temporary microburst from another flow). Consequently, as discussed in Chapter 2, TCP often performs far from optimal.

Several approaches along with PCC Allegro, e.g. PCP [23], and BBR [28] have been proposed for fixing traditional TCP congestion control's architectural deficiencies by adapting sending rates in response to aggregated, meaningful performance metrics like throughput, loss rate, and latency. BBR and PCP share a similar architecture and we use the more recently proposed BBR as example shown in Figure 4.1. While BBR gathers meaningful performance metrics and so improves deficiency (1) BBR, like TCP, does not learn a causal relationship between its rate-control actions and performance. Instead, BBR rate-control relies on assumptions about the network (e.g., very deep buffer, single competing CUBIC flow, stable latency) and hardwired change-patterns in performance (e.g., increase in delivery rate) to predefined reactions. Consequently, BBR's white-box approach inherits TCP's deficiency: its performance, convergence properties, and level of TCP-friendliness, suffer when the presumed network conditions do not match reality, e.g., under very high RTTs, dynamically changing network conditions, shallow queues, and more (shown in our evaluation in § 4.5).

To give a quick review (more details in Chapter 2), the gist of PCC's architecture is depicted in Figure 4.1. Time is divided into consecutive intervals, called Monitor Intervals (MIs), each devoted to "testing" the implications for performance of sending at a certain rate. Suppose that a PCC sender sends at rate r in some MI. PCC sender aggregates selective ACKs (SACKs) for packets sent in that MI into meaningful performance metrics such as throughput, loss rate and averaged latency, and feeds these into a utility function that translates these into a numerical value. PCC's rate-control module constantly adjusts the sending rate in the direction that is most beneficial in terms of utility. Putting it under the lens of online learning control principles, PCC's blackbox rate-control architecture both quantifies performance explicitly, and infers how rate changes impact performance.

Yet, PCC Allegro, as a specific realization of the PCC architecture, is suboptimal from an online learning perspective. In particular, PCC Allegro's simple rate control scheme decides whether the transmission rate should be increased or decreased based on empirically-derived utility, but does not take utility into account when determining the increase/decrease step size. We illustrate this point through a simple example. Consider a single link of capacity C. Suppose that PCC's utility function is as described in Figure 4.2, i.e., higher throughput is rewarded, but sending at a rate of above C leads to packet loss and so lowers the utility. Now, consider two possible initial rates for

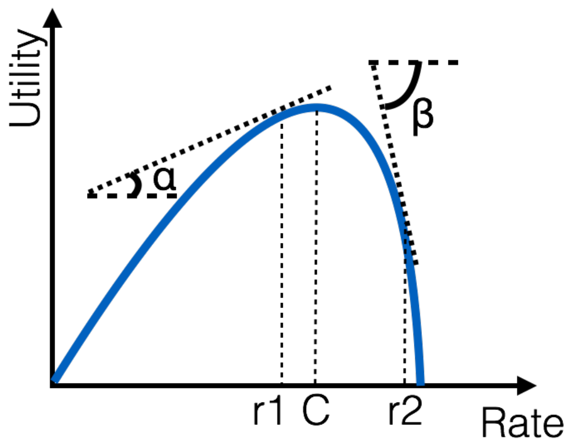


Figure 4.2: Utility-based rate-control

the PCC-sender: r_1, which is slightly smaller than C, and r_2, which is significantly higher than C. When the initial sending rate is r_1, the PCC-sender will increase its rate to $r_1(1+\epsilon)$, for a fixed $\epsilon > 0$, whereas when the initial rate is r_2, the sender will decrease its rate to $r_2(1-\epsilon)$. Intuitively, this rate-adjustment is not optimal; choosing a too small ϵ will result in lowering the rate too slowly from r_2, and thus taking too long to converge to the optimum utility, whereas choosing a too large ϵ will increase the rate from r_1 by too much, overshooting the optimum. Indeed, any choice of *fixed* increase/decrease step size is bound to be too much in some circumstances and too little in others, resulting in suboptimal reactivity and convergence[1]. Intuitively, PCC Allegro's suboptimal behavior suggests that the "right" action should take into account of the *amount* of utility value change proportionally. This motivates PCC Vivace's online learning rate-control.

Besides this suboptimal rate control algorithm, built into PCC Allegro is a somewhat "arbitrary" choice of utility function framework. Indeed, looking back, we designed that utility function based on intuition and simulations. While we prove that that specific choice induces desirable properties in specific settings, PCC Allegro's behavior at convergence state remains little understood. Specifically, while fair convergence is provably guaranteed when the utility function is not latency-aware, this is not so for latency-aware PCC. In addition, PCC Allegro's utility function framework makes it difficult to reason about fundamental tradeoffs in parameter settings, the interaction of heterogeneous PCC senders without resource isolation, and more.

[1]While PCC Allegro's step size automatically increases after multiple decisions to move in the same direction, PCC Allegro still inherently ignores the information reflected in the utility when deciding on step size.

The combination of PCC Allegro's sub-optimal rate control scheme and ad hoc choice of utility function prevents PCC Allegro from attaining good performance under rapidly changing network conditions, does not alleviate bufferbloat, results in as slow a convergence rate as TCP, leads to high packet-loss upon convergence, and is overly aggressive towards TCP. Hence, while using a promising architecture, PCC Allegro is still far from optimal.

Like PCC Allegro, PCC Vivace also tests different rates at consecutive Monitor Intervals (MIs), translates aggregate performance statistics into numerical utility values, and feeds this information into a rate-control algorithm to determine the next sending rates. However, PCC Vivace employs a principled online optimization approach to replace both fundamental components of the PCC architecture, namely: (1) the utility-derivation framework, and (2) the online rate-control algorithm. We identify a class of utility functions to build into PCC Vivace that provably guarantees near-optimal performance at equilibrium, sensitivity to latency (when desirable), improved friendliness to legacy TCP, and more (§ 4.3). PCC Vivace employs a gradient-ascent-inspired online learning rate-control scheme that is carefully engineered to achieve high performance, fast and stable convergence and quick reaction to changes in network conditions (§ 4.4). We show (§ 4.5) that PCC Allegro's embedded online optimization significantly outperforms both PCC Allegro and BBR under real-world network conditions.

4.3 PCC Vivace's Utility Framework

We present below PCC Vivace's utility-derivation framework, which draws ideas from results in online optimization about decision making in the face of uncertainty.

Like PCC Allegro, PCC Vivace divides time into consecutive Monitor Intervals (MIs). At the end of each MI, the Vivace-sender i applies the following utility function to transform the performance statistics gathered at that MI to a numerical utility value:

$$u(x_i) = x_i^t - bx_i\frac{d(RTT_i)}{dt} - cx_i \times L_i, \tag{4.1}$$

where x_i is sender i's sending rate and L_i is its observed loss rate. The term $\frac{d(RTT_i)}{dt}$ is the observed "RTT gradient" during this MI, *i.e.*, the increase in latency experienced within this MI. ($t, b, c \geq 0$

are weights.) Intuitively, utility functions of the above form reward increase in throughput (x_i^t), and penalize increase in latency ($bx_i\frac{d(RTT_i)}{dt}$) and increase in loss ($cx_i \times L_i$).

Observe that PCC Vivace's utility function does not consider the absolute value of latency, but, instead, uses "RTT gradient" to penalize increases in latency. To see why, consider the following example. A single sender on a link sends at a rate of twice the capacity of the link for a single MI and then tries a slightly lower, but still much higher than the capacity, rate in the consecutive MI. Such a sender would experience higher absolute latency in the second MI than in the first MI (since bufferbloat is only further aggravated), even though lowering the rate was clearly the right choice. To learn that lowering the rate is more beneficial, the sender must examine the rate at which latency increases/decreases.

The choice of values for the parameters b, c and t in the utility function has crucial implications for the existence of an equilibrium point when multiple PCC Vivace senders compete and for the performance level at such an equilibrium.

4.3.1 Stability and Fairness

When $t \leq 1$, this above choice of utility functions to build into PCC Vivace falls into the category of "socially-concave games" in game theory [37]. A utility function within this category, when coupled with PCC Vivace's online-learning rate-control scheme (described in § 4.4), guarantees high performance from the individual sender's perspective and ensures quick convergence to a global rate-configuration [44, 81].

Specifically, we consider a network model with a set N of n competing senders sharing a bottleneck link of capacity C and FIFO-queue. Under this model, $L = \max\{1 - \frac{C}{\sum_{j\in N} x_j}, 0\}$, and $\frac{d(RTT_i)}{dt} = \frac{\sum_{j\in N} x_j - C}{C}$. Hence, the utility function translates to:

$$u(x_i) = x_i^t - bx_i\frac{\sum_{j\in N} x_j - C}{C} - cx_i(1 - \frac{C}{\sum_{j\in N} x_j})$$

We apply ideas from online learning theory and game theory to prove Theorem 1, that is, multiple PCC Vivace senders converge to a unique and fair equilibrium point. One common question to ask is that how can we express the loss rate as $L = \max\{1 - \frac{C}{\sum_{j\in N} x_j}, 0\}$ and gradient of RTT as $\frac{d(RTT_i)}{dt} = \frac{\sum_{j\in N} x_j - C}{C}$, if a single sender does not have knowledge of other senders sending rate? We would like to highlight that this expression is only used in the process of analyzing the existence and uniqueness of Nash equilibrium. During the actual data transfer process, senders are observing these performance metrics entirely locally.

Theorem 1. *When N Vivace-senders share a single link of capacity C, each Vivace-sender i's utility function is $u_i = x_i^t - bx_i\frac{\sum_{j\in N} x_j - C}{C} - cx_i(1 - \frac{C}{\sum_{j\in N} x_j})$, and $t \leq 1$, convergence to a fixed configuration of sending rates $(x_1^*, \ldots, x_n^*)$ such that $x_1^* = x_2^* = \ldots = x_n^*$ is guaranteed.*

Assuming a single-bottleneck tail drop queue, we have $L = 1 - \frac{C}{S(X)} = 1 - \frac{C}{nx_i}$, where $S(X) = \sum_{j\in N} x_j$. And the change rate in the bottleneck queue size is $\sum_{j\in N} x_j - C$. In addition, since the rate of which the queue is drained is C, we have, $\frac{d(RTT_i)}{dt} = \frac{\sum_{j\in N} x_j - C}{C}$. Hence, the utility function can be described as,

$$u(x_i, L, RTT) = x_i^t - bx_i\frac{\sum_{j\in N} x_j - C}{C} - cx_i(1 - \frac{C}{S(X)})$$

Lemma 12. *There is a unique NE in this game.*

Proof. We now show that $u(x_i, L, RTT)$ is strictly socially concave.

1. Property 1 (The sum of utilities is concave in X).

$$\bar{x} = \sum_i \frac{1}{N} u_i(x) = \frac{1}{N}\sum_i u_i(x) \tag{4.2}$$

$$= \frac{1}{N}\sum_i \left(x_i^t - cx_i(1 - \frac{C}{S(X)}) - bx_i\frac{\sum_{j\in N} x_j - C}{C} \right) \tag{4.3}$$

Abusing notation we will denote $S(x)$ by S. To check concavity condition we take the second derivative,

$$\frac{d^2\bar{x}}{dS^2} = \frac{d^2}{dS^2}\left(\sum_i x_i^t - bS(1 - \frac{C}{S}) - cS\left(\frac{S - C}{C}\right) \right) \tag{4.4}$$

$$= a\frac{d^2 \sum_i x_i^t}{dS^2} - \frac{2b}{C} \tag{4.5}$$

$$\tag{4.6}$$

Since $t < 1$, x_i^t is a concave, and then $\sum_i x_i^t$ is concave, hence the first term in the utility function is concave . The second term in the utility function is concave since by (4.5), $-b\frac{2}{C} < 0$. Therefore, utility function is concave in $X = (x_1, ..., x_n)$.

2. Property 2 (convex in the others user's utilities). Let $r = \sum_i x_{-i}$. Then $S = x_i + r$. Since r is a linear function in x_{-i} it is enough to show convexity in r. We start by rewriting the utility as a function of r,

$$u(x_i, L, RTT) = x_i^t - bx_i \frac{x_i + r - C}{C} - cx_i(1 - \frac{C}{x_i + r})$$

$$\frac{d^2 u(x_i, r)}{dr^2} = cx_i \frac{(4S - 2C)\left(2 - \frac{C}{S}\right)}{S^2 (2S - C)^2} \tag{4.7}$$

Note that (4.7) is independent of b. Since $S = r + x_i > C$, we have $4S - 2C > 0$. In addition, $2 - \frac{C}{x+r} > 0$ (as $S > C$). Therefore $\frac{d^2 u(x_i, x_{-i})}{(dx_{-i})^2} > 0$ and then $u(x_i, x_{-i})$ is convex in x_{-i} □

On top of the stable and fair convergence guarantee, b and c determines additional properties of PCC Vivace's utility function as discussed below.

4.3.2 Convergence without Latency Inflation

Regarding the latency property at convergence state, the ideal case, in theory, would be there is no latency inflation beyond the base RTT. To achieve that, b needs to satisfy $b \geq t * N^(2 - t)C^(t - 1)$.

Lemma 13. *For any number of senders n, if $tn^{2-t}C^{t-1} \leq b$ then there is no packet loss*

Proof. First note that for sufficiently large penalty of the latency, the senders will not suffer loss. Hence, we seek for the minimal b so that once the latency increases the senders will lower their rate. Hence we take the derivative of all senders and then sum them up,

$$\sum_i \frac{du_i}{dx_i} = \sum_i \frac{d}{dx_i}\left(x_i^t - bx_i \frac{d(RTT_i)}{dt}\right) \tag{4.8}$$

$$= \sum_i tx_i^{t-1} - \frac{b\,(x_i + r - C)}{C} - \frac{bx_i}{C} \tag{4.9}$$

$$= \sum_i t(\frac{S}{n})^{t-1} - \frac{b\,(S - C)}{C} - \frac{bx_i}{C} \tag{4.10}$$

$$= \sum_i t(\frac{S}{n})^{t-1} - \frac{bS}{C} + b - \frac{bx_i}{C} \tag{4.11}$$

$$= \sum_i t(\frac{S}{n})^{t-1} - \frac{bS}{C} + b - \frac{b\frac{S}{n}}{C} \tag{4.12}$$

$$= atn^{2-t}S^{t-1} - \frac{nbS}{C} + nb - \frac{bS}{C} \tag{4.13}$$

$$= atn^{2-t}S^{t-1} - \frac{(n+1)bS}{C} + nb \tag{4.14}$$

Note that the derivative decreases with S. Hence the maximal gradient is given when $S = C$. Hence to get a lower bound on b, we will set $S = C$. In addition assume that $n \leq S$. Hence we get,

$$atn^{2-t}C^{t-1} - (n+1)b + nb < 0 \tag{4.15}$$

$$atn^{2-t}C^{t-1} - (n+1)b + nb \leq 0 \tag{4.16}$$

$$=> atn^{2-t}C^{t-1} \leq b \tag{4.17}$$

□

In practice, we can choose realistic N and C such as 1000 concurrent flows competing on a single 1000Mbps bottleneck link. However, it is critical to note that if the network buffer size small or the number of competing sender is very large, complete lossless transmission usually cannot be achieved due to noise introduced in the dynamic interaction of different senders and the loss penalty may also come in play.

4.3.3 Random Loss *vs.* Congestion

Analysis of the parameter c in the utility function reveals new insights into a fundamental tradeoff between tolerance of random packet-loss and packet-loss upon convergence. Packet loss that is not due to network congestion (resulting from lossy wireless links, port flaps on routers, unreliably long distance cabling, etc.), is a common phenomenon in today's Internet [28].

PCC Vivace's endurance of random packet-loss is configurable and reflected in the choice of parameter c in PCC Vivace's utility function. To simplify exposition, suppose that $a = t = 1$ and $b = 0$, so the utility function is purely "loss-based". Enduring random packet-loss rate of p, implies that the derivative of u satisfies:

$$\dot{u}(x) = 1 - cp > 0, \quad \text{and so:} \quad c < \frac{1}{p}$$

We can prove, however, that when n Vivace-senders share a link, and $n > c$, then in the unique equilibrium to which PCC Vivace is guaranteed to converge (Theorem 1), the loss rate experienced by each sender is:

$$L = \frac{n/c - 1}{n - 1}$$

Plugging in $c = \frac{1}{p}$ yields:

$$L = \frac{np - 1}{n - 1}$$

Hence, **enduring random packet loss is not a "free lunch"**. Observe that as n approaches infinity, the loss rate on convergence is effectively lower bounded by p. This has crucial implications: **withstanding $x\%$ random loss comes at the cost of suffering $x\%$ packet-loss upon convergence with a large number of senders**. Our experimental analyses of TCP, BBR, and PCC Allegro, suggest that the same tradeoff applies to these protocols as well, leading us to conjecture that this is a fundamental tradeoff in congestion control protocol design. This might suggest, e.g., that BBR's tolerance of 15% random packet-loss is too high and must be lowered to attain good performance upon convergence.

A General Case Proof

The above conclusion can be generalized to work with any loss-based ($b = 0$) utility function in our strictly socially concave utility function framework. We define that any protocol is *p-loss-endurable* if it does not decrease its rate under random loss rate p.

Lemma 14. *A sender is* p-loss-endurable *if* $c < \frac{tC^{t-1}}{p}$

Proof. If there is no congestion then $L = p$, and the utility function can be described as,

$$u(x_i, L) = x_i^t - cx_ip \tag{4.18}$$

We now check under what values of c, such that $n \to \infty$, we have $\sum_i \frac{du_i}{dx_i} > 0$.

$$\sum_i \frac{du_i}{dx_i} = \sum_i \frac{d}{dx_i}\left(x_i^t - cx_ip\right) \tag{4.19}$$

$$= \sum_i \left(tx_i^{t-1} - cp\right) \tag{4.20}$$

$$= n^{2-t}atS^{t-1} - ncp \tag{4.21}$$

What is the upper bound on c so that $\sum_i \frac{du_i}{dx_i} \geq 0$

$$n^{2-t}tS^{t-1} - ncp > 0 \tag{4.22}$$

$$=> n^{2-t}tS^{t-1} > ncp \tag{4.23}$$

$$=> c < \frac{n^{1-t}t}{S^{1-t}p} \tag{4.24}$$

The lower bound is given when $S = C$ (as the gradient increases with S) and $n = 1$ (by the inequality), in which case the maximal c is,

$$c = \frac{tC^{t-1}}{p} \tag{4.25}$$

□

Lemma 15. *Any p-loss-endurable under NE, satisfies the following,*

$$n^{1-t}p - \frac{L}{(1-L)^{1-t}} - \frac{1}{n} = 0 \tag{4.26}$$

Proof. In order to derive the loss rate at NE, we sum the derivative of all senders

$$\sum_i \frac{du_i}{dx_i} = tS^{t-1}n^{2-t} - c\left[n\left(2 - \frac{C}{S}\right) - n + \frac{C}{S}\right] \tag{4.27}$$

where NE is given when $\sum_i \frac{du_i}{dx_i} = 0$. By Lemma (14), To endure packet loss p we set $c = \frac{atC^{t-1}}{p}$, we have

$$=> tS^{t-1}n^{1-t} = \frac{tC^{t-1}}{p}\left[\left(1 - \frac{C}{S}\right) + \frac{C}{nS}\right] \tag{4.28}$$

$$=> \left(\frac{C}{S}\right)^{1-t} n^{1-t}p = \left(1 - \frac{C}{S}\right) + \frac{C}{S}\cdot\frac{1}{n} \tag{4.29}$$

$$=> (1-L)^{1-t}\, n^{1-t}p = (L) + (1-L)\frac{1}{n} \quad \text{using } \frac{C}{S} = 1 - L \tag{4.30}$$

$$=> n^{1-t}p = \frac{L}{(1-L)^{1-t}} + \frac{1}{n} \tag{4.31}$$

□

Using the above general conclusion, we can easily get the aforementioned conclusion as follow.

Corollary 2. *Under NE, if $t = 1$ then for $n \to \infty$, and no loss for $L = p$*

Proof. Plugging in the relation of Lemma 15 we get

$$L = \frac{np - 1}{n - 1} \tag{4.32}$$

which equal to p when $n \to \infty$. □

4.3.4 TCP Friendliness

A common requirement from new congestion control schemes is TCP friendliness, i.e., fairly sharing bandwidth with existing TCP connections (e.g., CUBIC). However, attaining perfect friendliness to TCP can be at odds with achieving high performance, as the congestion control protocol is expected to both not aggressively take over spare capacity freed by a TCP connection when it backs off, and quickly take over spare capacity freed by the very same TCP connection when it terminates. We conjecture that it is fundamentally hard for any loss-based protocol to achieve consistently high performance and at the same time be fair towards TCP. The best is to hope it does not dominate TCP too much.

Worse yet, latency-aware protocols can be entirely dominated by today's prevalent loss-based TCP CUBIC. Fast TCP [75], for instance, backs off as latency deviates from the minimal latency due to TCP CUBIC continuously filling the network buffer.

How, then, can a rate-control protocol both optimize latency and avoid being killed by loss-based TCP connections? We argue that PCC Vivace's utility function, which does not rely on absolute latency but on the above described "latency gradient" is a big step in this direction. Informally, PCC Vivace's utility function captures the objective that can be expressed as "care about latency when your rate selection makes a difference". To see this, consider the scenario that a Vivace-sender is the only sender on a certain link, it tries out two rates that exceed the link's bandwidth, and the buffer for that link is not yet full. PCC Vivace's utility function will assign a higher value to the lower of these rates, since the achieved goodput and loss rate are identical to those attained when sending at the higher rate, but the latency gradient is lower. Thus, in this context, the Vivace-sender behaves in a latency-sensitive manner and reduces its transmission rate. Now, consider the scenario that the Vivace-sender is sharing a link that is already heavily utilized by many loss-based protocols like TCP CUBIC and the buffer is, consequently, almost always full. When testing different rates, the Vivace-sender will constantly perceive the latency gradient as roughly 0, and thus disregard latency and compete against the TCP senders over the link capacity, effectively transforming into a loss-based protocol. Our experimental results in § 4.5.3 illustrate this intuition.

4.3.5 Practical Utilities

In light of the above theoretical insights, our implementation of PCC Vivace employs the following default utility function:

$$u(x_i) = x_i^{0.9} - 11330x_i\frac{d(RTT_i)}{dt} - 11.35x_iL_i \tag{4.33}$$

This function is carefully chosen to endure 5% random packet-loss rate (c) and theoretically achieve no inflation in latency with up to 1000 competing senders on a 1000Mbps bottleneck link (b). Importantly, this choice of utility function (as also experimentally shown in § 4.5) induces improved friendliness towards TCP.

4.3.6 Heterogeneous Senders

So far, our discussion focused on the environment where all Vivace-senders employ the same utility functions (i.e., Vivace-senders are homogeneous). However, our utility functions framework allows us to reason about the outcome of interactions between heterogeneous Vivace-senders competing over shared bandwidth resources. PCC Allegro also implied this possibility, but did not give meaningful insights due to its arbitrarily chosen utility function framework. This potentially has far reaching implications, as discussed below.

Recent studies on SDN-based traffic engineering [45, 49] and network optimization for big-data systems [30] suggest a need for resource allocation at the transport layer. However, globally allocating network resources to transport-layer connections usually involves complex schemes for rate-limiting at end-hosts, or utilizing in-network isolation mechanisms. OpenTCP [40] proposes allocating bandwidth by tuning TCP parameters or switching between TCP variants. Yet, as TCP has no direct control knobs for global network-resource allocation, OpenTCP resorts to clever "hacks" and complicated feedback loops to indirectly achieve such control.

PCC Vivace's utility function (game-theoretic) framework, in contrast, provides great flexibility in resource-allocation. For concrete example, consider the following loss-based utility:

$$u_i(x) = x_i - c_ix_i\left(\frac{1}{1-L} - 1\right)$$

This utility function, similarly to the utility function in § 4.3, induces a unique equilibrium point to which PCC Vivace senders are guaranteed to converge. Now, suppose that n Vivace-senders share a link and the goal is to allocate the link's bandwidth C between the senders by assigning a rate of x_i to each sender such that $\sum_{j \in N} x_j = C$. We next prove, by solving the equilibrium, that this exact bandwidth allocation is achievable by setting $c_i = \frac{C}{x_i}$ in every u_i. We experimentally validate this result in § 4.5. Hence, through tuning a single parameter in the utility function, Vivace can flexibly allocate bandwidth to competing senders.

Proof of Vivace's Tunable Nash Equilibrium

$$u_i(x, L) = x_i - c_i x_i \left(\frac{1}{1-L} - 1 \right)$$

This translate to

$$u_i(x, L) = x_i - c_i x_i \frac{S}{C} + c_i x_i$$

Hence,

$$\frac{du_j}{dx_i} = \frac{d}{dx_i} \left(x_i - c_i x_i \frac{S}{C} + c_i x_i \right) \tag{4.34}$$

$$= \frac{d}{dx_i} \left(x_i - c_i x_i \frac{x_i + r}{C} + c_i x_i \right) \tag{4.35}$$

$$= 1 - \frac{2c_i x_i}{C} - \frac{c_i \sum_{j \neq i} x_j}{C} + c_i \tag{4.36}$$

$$\tag{4.37}$$

To get the c_i in NE we check first optimality condition,

$$1 - \frac{2c_i x_i}{C} - \frac{c_i \sum_{j \neq i} x_j}{C} + c_i = 0 => 1 = \frac{c_i x_i}{C}$$

Therefore, we got

$$c_i = \frac{C}{x_i}$$

4.4 PCC Vivace's Rate Control

PCC Vivace's rate-control is divided into two phases: slow-start and online learning. A Vivace-sender exits the slow-start phase when its empirically-derived utility value decreases for the first time and, at that point, enters the online learning phase and never again returns to slow-start.

4.4.1 Key Idea

PCC Vivace's online learning phase employs an online gradient-ascent learning scheme to select transmission rates. This choice of rate-control algorithm is very appealing from an online optimization theory perspective [37, 44, 81]. Specifically, when the utility functions are strictly convex, which is satisfied when $t < 1$ in our utility function formulation (Equation 4.1), the following two desiderata are fulfilled. (1) Each sender is guaranteed that employing PCC Vivace is (asymptotically) no worse than the optimal *fixed* sending rate in hindsight, termed the "no-regret" guarantee in online learning literature [37, 44, 81]. Note that this applies even with other senders simultaneously adjusting their rates and regardless of other senders' rate-control schemes and the network conditions. (2) When multiple senders share the same link, quick convergence to an equilibrium point is guaranteed.

To provide some more details for the above two claims, we leverage the following important theoretical results. Zinkevich [82] proves that gradient descent algorithm, defined over convex utility function with bounded derivative, satisfies the no-regret property. Even-Dar et al. [36] show that in socially concave games the average action of no-regret algorithm converge to the a Nash equilibrium of the game. Strictly socially concave games is a subclass of socially concave games where its utility function satisfies one of the following

1. strict concavity of the sum of all players' utilities.

2. strict convexity in the other players strategies.

3. strict convexity in the players' own strategy.

In addition, strictly socially concave utility function guarantees unique Nash equilibrium and no-regret algorithms converge to the unique NE of the game. Since PCC Vivace's rate control algorithm is based on gradient descent, in order to prove convergence we only need to prove that the game is strictly socially concave. We have proved that in § 4.3.1. In addition, since all utility functions' senders are equivalent, the unique NE is symmetric and fair. Therefore, from any initial rate configuration the players converge to a fair state in which all senders send at the same rate.

4.4.2 Practical Challenges

Realizing gradient ascent schemes from online learning literature in practice involves highly non-trivial operational challenges. In theory, applying gradient ascent to rate-control means starting at some initial transmission rate and repeatedly estimating the gradient of the utility function γ through sampling and changing the rate by $\theta\gamma$, where θ is initially set to be a very high positive number. With time, θ gets gradually smaller, approaching 0.

The first challenge in realizing this approach is estimating the gradient of the utility function. What happens when the environment is noisy, e.g., due to complex interactions between multiple senders or to inherent randomness (e.g., lossy wireless links)? Also, when sending at low rates, the performance-related statistics gathered by a PCC Vivace-sender can be highly unreliable as the sampling rate and duration are inadequate. Another major challenge is deciding on the extent to which the rate should be increased/decreased. Specifically, adhering to the above theoretical rate-adjustment rule suffers from two serious problems: (1) Initially, the increase or decrease step size is potentially huge, which can result in a sender jumping between very low rates (e.g. 1Mbps) and very high rates (e.g. 500Mbps) in a matter of tens of milliseconds. Such high rate variance can result in high loss rates and latency inflation; (2) As time goes by, and θ diminishes, changes in rate become increasingly small, leading to very slow reactivity to changes in network conditions (e.g., freed capacity, network congestion). These challenges, unless carefully addressed, void the practical benefits of applying gradient-ascent-based approaches to congestion control. We next explain how this is tackled in PCC Vivace.

4.4.3 Translating Utility-Gradients to Rates

PCC Vivace's online learning algorithm computes the gradient of the utility function by sampling two rates, above and below the current rate, and quantifying the difference, in terms of utility, as follows. Suppose that the current sending rate is r. Then, in the next two MIs, the sender will test the rates $r(1+\epsilon)$ and $r(1-\epsilon)$, compute the corresponding numerical utility values, u_1 and u_2, respectively, and estimate the gradient of the utility function to be $\gamma = \frac{u_1-u_2}{2\epsilon r}$. Then, PCC Vivace utilizes γ to deduce the direction and extent to which rate should be changed, selects the newly computed rate, and repeats the above process.

To convert γ into a change in rate, PCC Vivace starts with fairly low "conversion factor" and increases the conversion factor value as it gains confidence in its decisions. Specifically, initially θ is set to be a conservatively small number θ_0 and so, at first, the rate change is $\Delta_r = \theta_0 \times \gamma$. We introduce the concept of *confidence amplifier*. Intuitively, when the sender repeatedly decides to change the rate in the same direction (increase vs. decrease), the confidence amplifier is increased. The confidence amplifier is a monotonically nondecreasing function that assigns a real value $m(\tau)$ to any integer $\tau \geq 0$. After a Vivace-sender makes τ consecutive decisions to change the rate in the same direction, θ is set to $m(\tau) \times \theta_0$ (and so the rate is changed by $\Delta_r = m(\tau) \times \theta_0 \times \gamma$). Setting $m(0) = 1$ implies that initially the change in rate is $\theta_0 \times \gamma$, as described above. When the direction at which the rate is adapted is reversed (from increase to decrease or vice versa), τ is set back to 0 (and the above process starts anew).

Sampled utility-gradient can be excessively high due to unreliable measurements or drastic changes to network conditions in between MIs. For instance, a burst of packet losses when probing a lower rate and no packet losses when probing a higher rate might result in the utility-gradient being huge and, consequently, a drastic rate change that overshoots the link's capacity. To address this, we introduce a mechanism, called *dynamic change boundary* ω. Whenever PCC Vivace's computed rate change (Δ_r) exceeds $\omega * r$, the effective rate change is capped at $\omega * r$. Informally, the dynamic change boundary is initialized to some predetermined value $\omega = \omega_0$, is gradually increased every time $\Delta_r > \omega$, and is decreased when $\Delta_r < \omega$. Specifically, ω is updated to $\omega = \omega_0 + k * \delta$ following k consecutive rate adjustments in which the gradient-based rate-change Δ_r exceeded the dynamic change boundary, for a predetermined constant $\delta > 0$. Whenever $\Delta_r \leq r * \omega$, PCC Vivace recalibrates the value of k in the formula $\omega = \omega_0 + k * \delta$ to be the smallest value for which $\Delta_r \leq r\omega$. k is reset to 0 when the direction of rate adjustment changes (e.g., from increase to decrease).

4.4.4 Contending with Unreliable Statistics

The conflicting goals of accurate measurements and fast reaction pose a significant challenge to PCC Allegro's rate control design; accurate measurements require long statistic accumulation time to prune out random noises, yet spending too much time measuring performance leads to slow reactions to changing network conditions. We next discuss the ideas incorporated into PCC Vivace's design to address this challenge.

Estimating the RTT gradient via linear regression. Computing the "RTT gradient" in MI x to express $\frac{d(RTT_x)}{dt}$ can be accomplished by quantifying the RTT experienced by the first packet and the last packet sent in that MI. Of course, such measurements can be very noisy. To well-estimate the RTT gradient with fairly few samples, we utilize linear regression. Specifically, PCC Vivace plots the 2-dimensional graph with the y-axis representing sampled RTTs and the x-axis representing the time of sampling, and uses the linear-regression-generated slope (the "β coefficient") as the RTT-gradient.

Low-pass filtering of RTT gradient. Non-congestion-induced latency jitters often occur in the real world, e.g., because of edge-router processing speed limits, recovery from packet losses in the physical layer (especially on wireless links), forwarding path flaps, or simply measurement errors due to processing time at the end-host networking stack. When PCC Vivace employs a latency-sensitive utility function, this can result in misinformed decisions. To resolve this, PCC Vivace leverages a low-pass filtering mechanism so as to ignore fast and small jitter in latency measurements (which falls under a predetermined threshold of $flt_{latency}$).

"Double checking" abnormal measurements. Occasionally, PCC Vivace's utility-gradient measurements lead to "counterintuitive" observations, e.g., sending at a higher rate results in lower loss rate. To determine whether these measurements reflect real network conditions (e.g., the above phenomenon can be explained by a reduction in rate by a competing sender), PCC Vivace "double checks" such counter-intuitive measurement results by re-running the same (pair of) micro-experiments. When the results of the double check are consistent with the original measurement results, PCC Vivace will average the utility-gradients computed in both sets of experiments and feed it into the rate-control algorithm.

Boost sampling at low rate. PCC Vivace relies on SACK feedback to compute performance metrics. Yet, when the sending rate is very low, only few packets are sent per MI. For example, under a 5Mbps rate on a 30ms RTT link, only 15 packets are sent in an RTT. Such a small number of samples is insufficient to reliably measure performance metrics such as loss rate, especially when network condition is very noisy. Hence, in such scenarios PCC Vivace optionally sends smaller

abnormality double-check	ON
low rate sampling boost	OFF
sampling step ϵ	0.05
initial conversion factor θ_0	1
initial dynamic boundary ω_0	0.05
dynamic boundary increment δ	0.1
RTT gradient filter threshold $flt_{latency}$	0.01
MI timeout $T_{timeout}$	5RTT
slow-start loss ignorance th_{loss}	0
slow-start latency inflation ignorance $th_{latency}$	0
confidence amplifier $m(\tau)$	$\tau \quad (\tau \leq 3)$ $2\tau - 3 \quad (\tau > 3)$

Table 4.1: PCC Allegro's rate control default parameters

packets so as to increase the number of samples.

Ignoring negligible packet loss and latency inflation in slow-start. Unreliable measurements are particularly harmful during slow-start when the rate is low. Slow-start is intended to quickly take advantage of available bandwidth and so exiting slow-start prematurely, e.g., due to inaccurate measurements.PCC Vivace optionally ignores very low packet loss rates and latency inflation (beneath the thresholds of th_{loss} and $th_{latency}$) during slow-start so as to not terminate the slow-start threshold as a result of low random loss or small measurement errors.

MI timeout. Generally, all information regarding packets sent out during an MI will be returned after approximately one RTT. However, in the case of sudden network condition changes, a large number of packets can be lost or delayed. The measurements PCC Allegro did before the sudden change are no longer meaningful. Therefore, when PCC Vivace does not learn the fate of all packets sent in an MI until a certain timeout $T_{timeout}$ (certain number of RTTs) is exceeded, PCC Vivace halves the sending rate. This is very useful in highly dynamic network conditions such as LTE environments, where network feedback might be too outdated to be useful.

Finally, we would like to point out that increasing the confidence of measurement data by intelligently filtering is a general challenge and many other methods are possible for future research.

PCC Vivace's online learning rate-control is designed to be robust and its consistent high performance is not highly sensitive to parameter choices. Different assignments for the parameters mainly affect the reactivity vs. stability tradeoff. We present all the parameters and their default

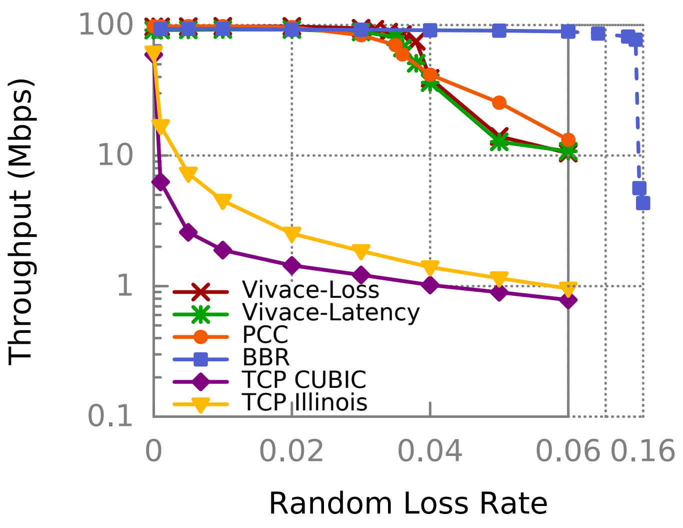


Figure 4.3: Random loss resistance

configuration setting in Table 4.1.

4.5 Evaluation

We implement a user-space prototype of PCC Vivace, and perform comprehensive experiments under emulated realistic network conditions[2] and on real-life Internet and applications to evaluate PCC Vivace. Unless otherwise specified, we use the default utility function described in § 4.3.5, and rate-control parameters listed in Table 4.1. To cleanly separate Vivace's loss-related properties from its latency-related properties, we further denote PCC Vivace as "*Vivace-loss*" when latency penalty coefficient $b = 0$ and and as "*Vivace-latency*" when $b = 900$ (the default).

4.5.1 Consistent High Performance

Resilience to Random Loss

Using Emulab [76], we evaluate the throughput of PCC Vivace on a 100Mbps bandwidth, 30ms RTT and 75KB buffer link with varying random loss rate, and compare it with PCC Allegro, BBR, and two TCP variants. As shown in Figure 4.3, both Vivace-Loss and Vivace-Latency achieve more than 90Mbps throughput when the random loss rate is at most 3%, and remain above 80Mbps until 3.5% loss rate. After that point, corresponding to the employed 5% loss resistance in the

[2]using Emulab [76] and Mahimahi [61]

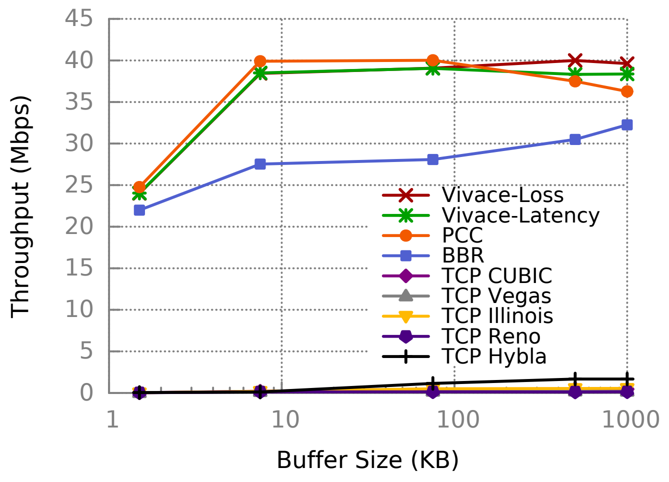


Figure 4.4: Long RTT tolerance

utility functions, their throughput reduces to $\frac{1}{10}$ of link capacity in that case. PCC Vivace does not achieve full capacity at close to 4% random loss (about 40%) due to temporary bursty losses, which may exceed 5% in some monitor intervals. Comparing to PCC Allegro with the same theoretical loss tolerance, the cut-off behavior of PCC Vivace is cleaner thanks to its new utility function framework. BBR keeps close-to-capacity throughput until 15% loss, because it is designed with such high loss resilience. By tuning PCC Vivace's utility parameter c, we can achieve similarly high resilience to random loss. However, the theoretical insights (§ 4.3.3) and experiments we later present (§ 4.5.2) suggest that 15% is not an reasonable design choice. Finally, Figure 4.3 also shows gains of 20∼50× over TCP family protocols.

High Performance on Satellite Links

We set up an emulated satellite link (as done in [32]) with 42Mbps, 800ms RTT and 0.74% random loss. Figure 4.4 shows the throughput achieved by different protocols. Both Vivace-Loss and Vivace-Latency perform at least similar to PCC Allegro, outperforming all the other protocols. Specifically, PCC Vivace reaches more than 90% link capacity with a 7.5KB buffer, in which case it is more than 40% larger than BBR. When the buffer size increases to 1000KB, the two PCC Vivace flavors are at least 20% better than BBR as well. We also observed $20 \sim 300\times$ higher performance comparing to the best-in-class TCP variant.

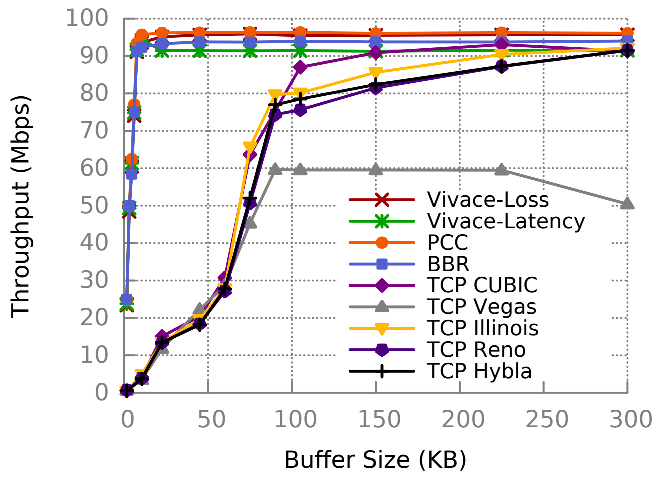


Figure 4.5: Vivace can achieve high throughput with small buffer

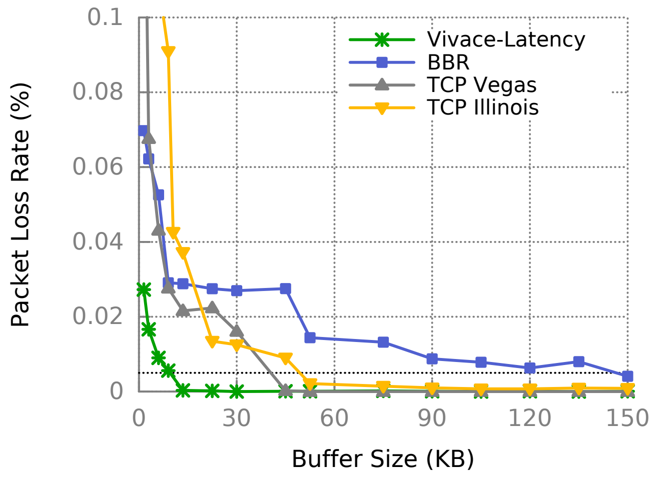


Figure 4.6: Vivace achieves low packet loss with shallow buffer

High Throughput without Bufferbloat

Existing loss-based TCP variants cannot utilize full bandwidth when the network buffer is shallow, and cause buffer bloat when network buffer is deep. PCC Allegro's default utility function is loss-based, so it achieves high bandwidth utilization, but still inflates the latency when buffer is deep. As PCC Vivace introduced a latency-aware and *provably fair* utility function framework, we evaluate (using Emulab) its throughput and latency performance on a 100Mbps bottleneck bandwidth and 30ms RTT link with different choices of network buffer. On the throughput front, as shown in Figure 4.5, to achieve more than 90Mbps, PCC Vivace, BBR and PCC Allegro only need a queue

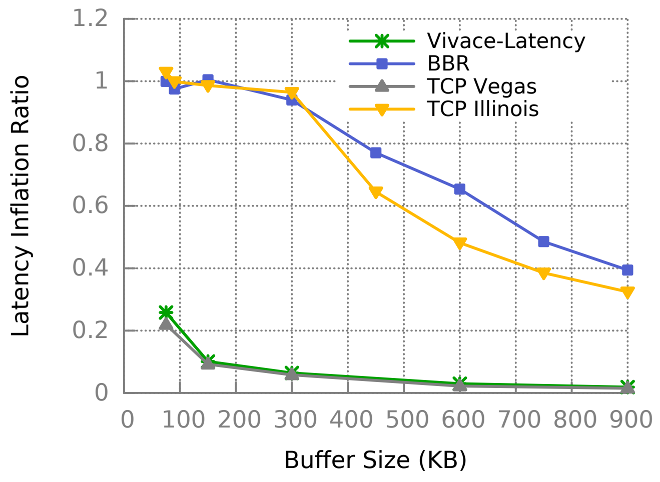
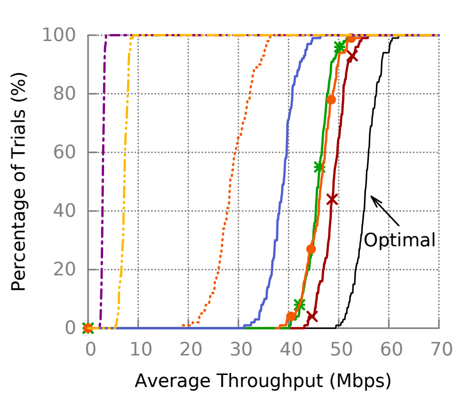


Figure 4.7: Vivace achieves negligible RTT overflow with shallow buffer

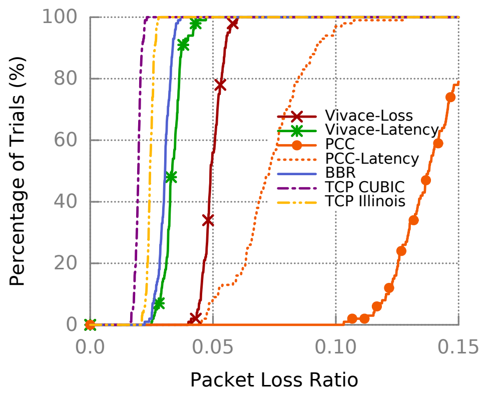
as shallow as 7.5KB, which is 95% smaller than needed by CUBIC.

We next focus on latency-aware protocols and compare the degree of latency inflation between Vivace-Latency, BBR and several latency based TCP variants. First, we study how small a buffer each protocol requires to achieve minimal latency inflation and near-lossless data transfer (less than 0.5% loss rate). As shown in Figure 4.6, TCP Vegas and Illinois need at least 45KB and 52.5KB buffer to have below 0.5% loss rate. BBR's packet loss rate keeps above 0.5% until the buffer size exceeds 150KB. As for Vivace-Latency, a 13.5KB buffer is already enough to guarantee nearly zero loss, which is 70%, 74.3%, and 91% smaller than that of Vegas, Illinois, and BBR, respectively.

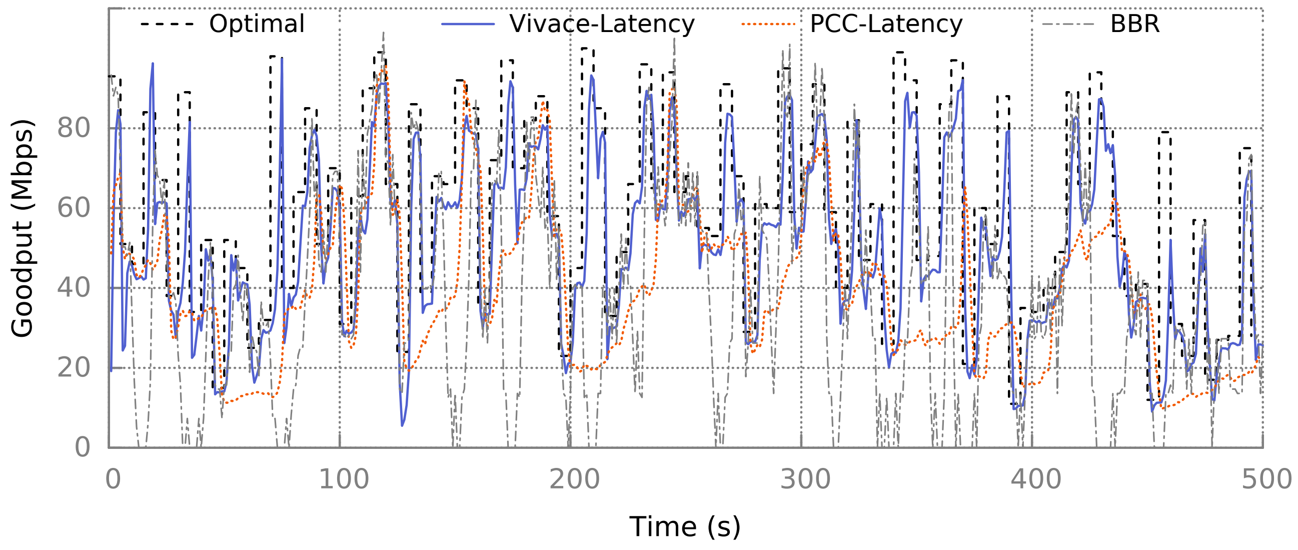
Second, we compare the latency inflation ratio, computed as the 95-th percentile self-inflicted RTT divided by the maximal possible latency inflation with given buffer size (when the buffer is full), in Figure 4.7. PCC Vivace's latency inflation ratio is kept small with its absolute RTT overflow always below 2ms. However, both TCP Illinois and BBR have close to 100% inflation ratio for as large as 300KB buffer, corresponding to almost full buffer. Even when the buffer size is 2BDP (750KB), Vivace-Latency still has a more than 90% less latency inflation ratio than BBR. We believe BBR's performance disadvantage is due to its white-box assumptions about the size of buffer in the network. Vegas achieves good performance on latency inflation by sacrificing the ability to fully utilize the available bandwidth (as shown in Figure 4.5). In sum, PCC Vivace achieves superior latency-awareness and high throughput at the same time.



(a) Achieving high capacity



(b) Moderate packet loss



(c) More responsive latency-sensitivity

Figure 4.8: PCC Vivace can adapt to rapidly changing network conditions

Swift Reaction to Changing Networks

We next demonstrate how PCC Vivace's online learning rate control significantly improves the reactiveness to dynamically changing network conditions.

Emulated changing networks. We start with an emulated changing network on Emulab where the RTT, bottleneck bandwidth, and random loss rate all change every 5 seconds with uniform distribution ranging from 10∼100ms, 10∼100Mbps, and 0∼1%, respectively. For each protocol, we repeat the experiment 100 times with 500s duration each, and calculate the cumulative distribution of average throughput and packet loss rate. Note that PCC Allegro's latency-based utility function, which does not guarantee fairness and convergence, is also evaluated, denoted by Allegro-Latency.

As shown in Figure 4.8(a), Vivace-Loss achieves the highest average throughput. Quantitatively, it reaches 49Mbps in median case, which is 88.3% of the optimal, corresponding to a gain of 5.4%, 25.6%, 72.5%, 5.7×, and 15.3× compared with PCC Allegro, BBR, Allegro-Latency, TCP Illinois, and CUBIC, respectively. Vivace-Latency performs similarly to PCC Allegro, still with a median gain of 17.9%, 62.0%, 5.3×, and 14.3× over BBR, Allegro-Latency, TCP Illinois, and CUBIC.

To further demonstrate PCC Vivace's reactivity, we compare different protocols' packet loss. As shown in Figure 4.8(b), the median case loss rates of Vivace-Loss and Vivace-Latency are only 4.9% and 3.3%. Specifically, Vivace-Latency has similar median loss as BBR (but higher throughput), while outperforming PCC Allegro and Allegro-Latency by 55.7% and 75.8%. This is because PCC Allegro's fix-rate control algorithm reduces rate too slowly when available bandwidth suddenly decreases. We also show a specific temporal trace in Figure 4.8(c). For clarity, we only show the results of Vivace-Latency, Allegro-Latency, and BBR. The figure shows that BBR occasionally suffers from sudden rate degradation. We find such situations happen because varying latency mismatches BBR's design assumption embedded in its hardwired rate control. Vivace-Latency, with its provably fair latency-based utility function, has significantly better reactivity compared with Allegro-Latency.

LTE networks. An even more challenging dynamic network scenario, as suggested by [78, 80], is the LTE environment where very deep queues are accompanied by drastically changing available bandwidth in a matter of milliseconds. On one hand, this extremely dynamic environment requires a long measurement time to prune out random noises. On the other hand, if Vivace takes too long to measure, the network condition may have drastically changed and invalidates any previous measurements. To improve reactivity and boost performance measurement, we reduce the MI timeout to $T_{timeout}$ = 2RTT, use a smaller MSS size of 500B and limit the maximal MI to 30 packets. In addition, because the LTE network provides flow isolation, we reduced the latency penalty coefficient b in utility function to 2.[3]

We use Mahimahi [61] to replay the Verizon-LTE trace provided by [78]. We compare PCC Vivace with Allegro-Latency, BBR, CUBIC, Vegas and Sprout [78]. Figure 4.9 shows the achieved

[3]In practice, the server often can distinguish a cellular connection by, for example, IP address and carrier analysis. Thus, it is able to tune the parameters accordingly.

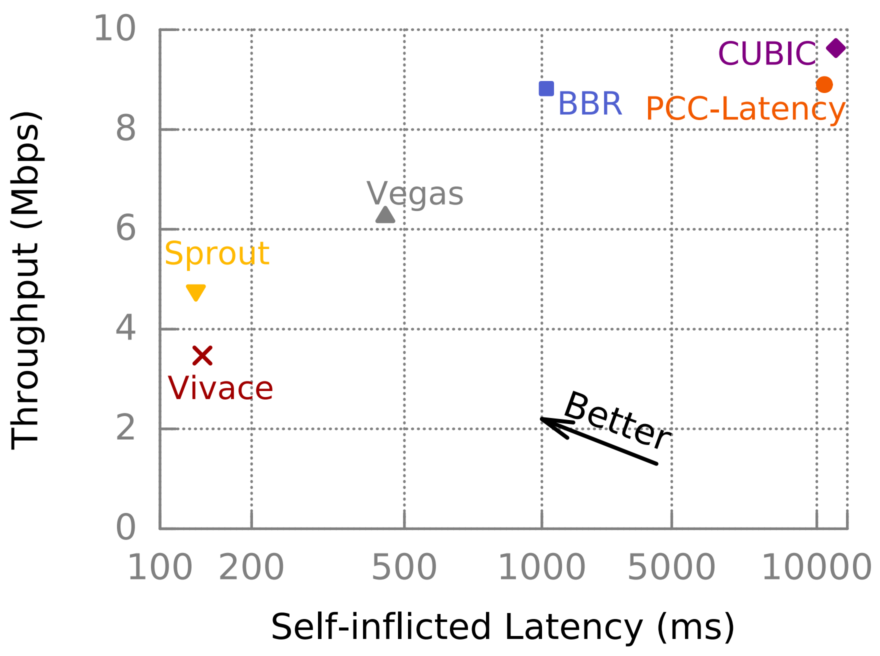


Figure 4.9: LTE throughput *vs.* self-inflicted latency

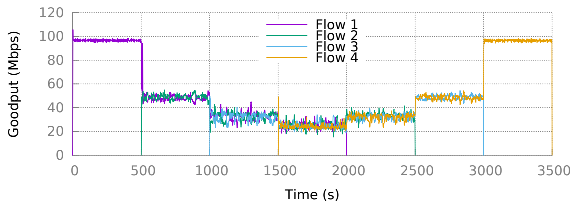
tradeoff between throughput and self-inflicted latency (as defined in [78]). PCC Vivace reduces latency by 98.8%, 87.2% and 97% compared to CUBIC, BBR and Allegro-Latency. Best-in-class TCP (Vegas) achieves 80% more throughput but 2.2× longer latency than PCC Vivace, and we believe this is a worse trade-off point than PCC Vivace. Moreover, TCP Vegas has other problems such as sub-optimal utilization of network resources and fragility against non-congestion loss.

Sprout performs better than PCC Vivace with 36.9% higher throughput and 4.8% less latency inflation. However, instead of the general non-isolated competing senders cases, Sprout is specifically designed for cellular networks with explicit measurement model of the cellular link, and requires receiver-side changes for explicit bandwidth feedbacks. We therefore believe that PCC Vivace, performing closely to Sprout without making assumptions about the network, provides further validation for PCC Vivace's online learning framework.

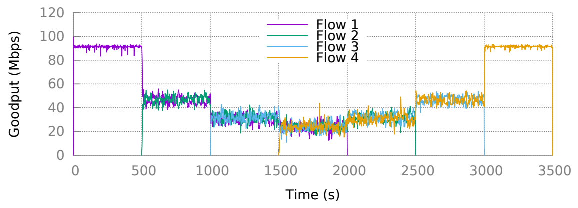
4.5.2 Convergence Properties

We next demonstrate that PCC Vivace improves convergence speed and stability tradeoff comparing to state-of-art protocols. We also experimentally show the important trade-off between congestion loss and random loss endurance.

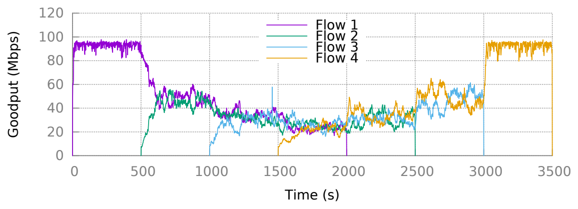
Better Convergence Speed and Stability



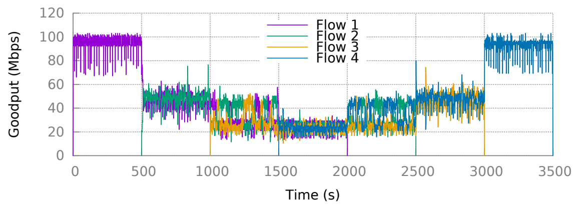
(a) Vivace-Loss



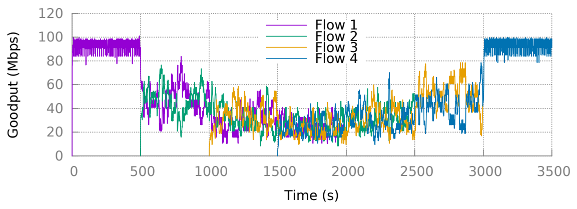
(b) Vivace-Latency



(c) Allegro-Latency



(d) BBR



(e) TCP CUBIC

Figure 4.10: Convergence Process Illustration

Temporal behavior of convergence. We set up a dumbbell topology on Emulab to demonstrate convergence performance with 4 flows sharing a 100Mbps bandwidth, 30ms RTT, 75KB buffer bottleneck link. Figure 4.10 shows the convergence process of several protocols with 1s granularity. It is visually apparent that PCC Vivace achieves fair rate convergence among competing flows and is more stable than BBR and CUBIC. Compared with Allegro-Latency, which does not have any convergence guarantee, PCC Vivace's default latency-aware utility function achieves significantly better convergence speed and stability at the same time.

Better convergence speed-stability tradeoff. We measure the quantitative trade-off between speed and stability of convergence using the same metrics as in [32]. With two competing flows on a 100Mbps, 30ms bottleneck link, we calculate the convergence time as the earliest point in time after which the second flow maintains a sending rate within ±25% of its ideal fair share (50Mbps) for at least 5s. The convergence stability is calculated as the throughput standard deviation of

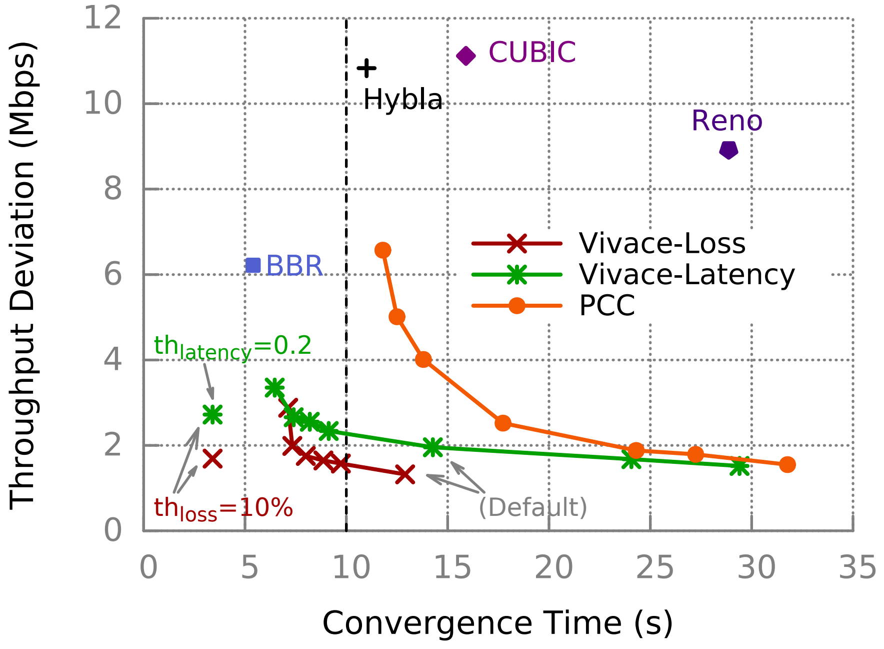


Figure 4.11: Better convergence speed and stability tradeoff curve

the second flow after its convergence. We reproduce the experiment results in [32] and plot them in Figure 4.11, these results illustrate that there is a “virtual wall” in the trade-off plane that neither TCP variants nor PCC Allegro can pass even by trading high variance at convergence state. Interestingly, BBR and PCC Vivace penetrate through that “virtual wall”. PCC Vivace, by default, positions at a significantly better tradeoff point. By assigning different values to the parameters in the control algorithm, we can generate a trade-off curve that is significantly better comparing to PCC Allegro’s similarly generated curve. Both Vivace-Latency and Vivace-Loss achieve similar convergence stability as PCC Allegro, but using nearly 60% smaller convergence time. With similar convergence speed of BBR, both PCC Vivace variants have around 50% smaller throughput deviation.

Furthermore, when a small amount of overhead is acceptable, PCC Vivace turns on the slow start packet loss and RTT inflation ignorance to $th_{loss} = 10\%$ and $th_{latency} = 0.2$. With this configuration PCC Vivace achieves both 37% faster and more stable convergence comparing to BBR.

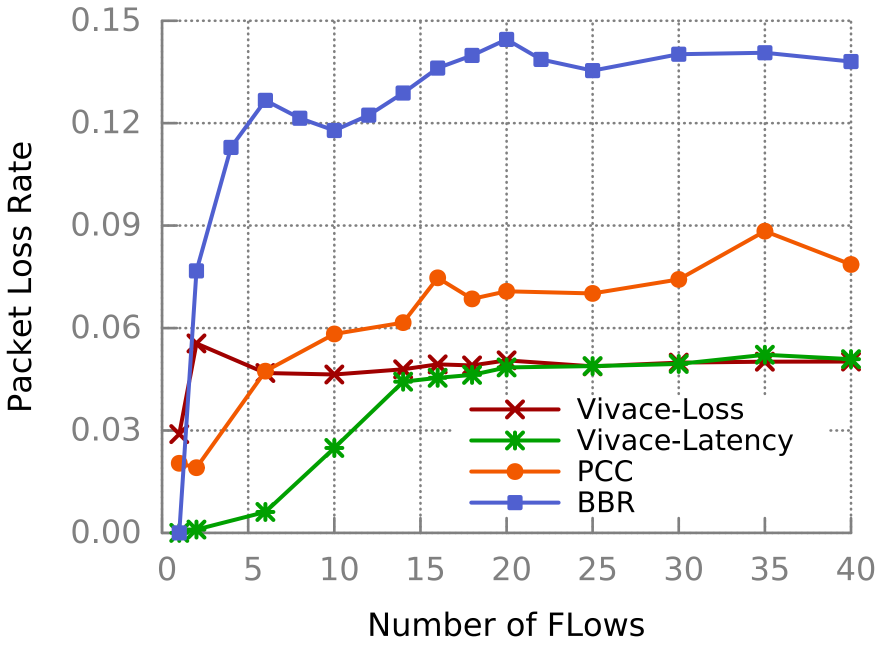


Figure 4.12: Convergence Congestion loss

The Right Price for Loss Resilience

As we analyzed in § 4.3.3, resilience to random loss comes at the cost of sustaining packet-loss after convergence when the number of senders increases. Importantly, this is only the theoretical "minimal price" one has to pay to endure random packet loss. Due the network dynamics from multi-sender interaction and efficiency of rate control algorithm, the actual price can be even higher. In this section, we experimentally evaluate this trade-off and demonstrate that PCC Vivace is positioned in a more favorable trade-off point than BBR and in practice, pays a price much closer to theoretical minimum than PCC Allegro. We setup an experiment with increasing number of concurrent competing flows with 30ms RTT. As we increase the number of flows, we proportionally increase the FIFO queue link's total bandwidth to maintain a per flow 8Mbps and 25KB (close to 1BDP) buffer share on average. Figure 4.12 shows the average packet loss each flow suffers as the number of concurrently competing flow increases.

We observe that the packet loss rate of Vivace-Loss converges at the theoretical bound of 5%. Even though BBR does not fall into PCC Vivace's online learning analysis framework, it shows a surprisingly similar tradeoff : it endures more random loss but pays a much higher price (14%

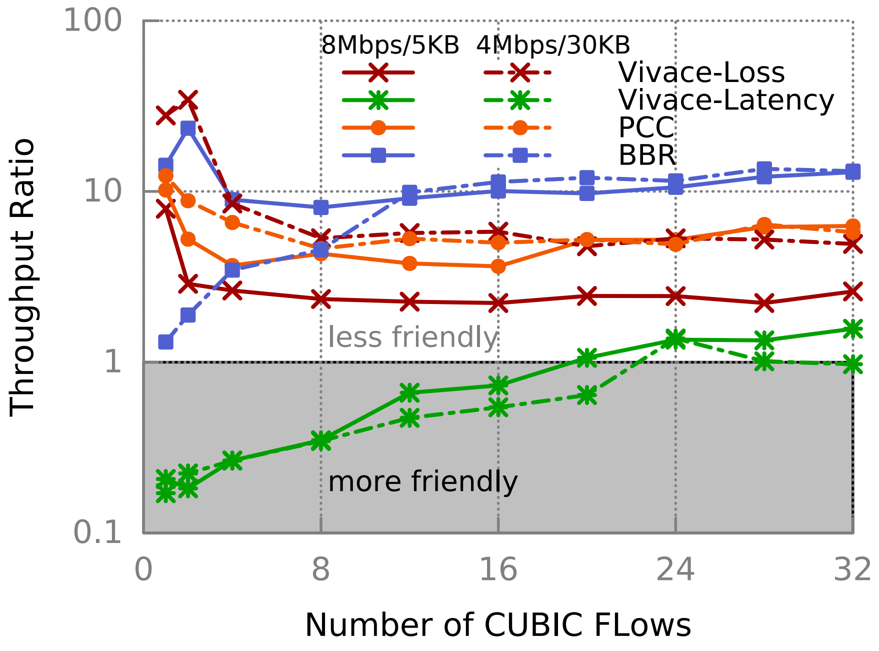


Figure 4.13: Vivace is a viable solution to TCP friendlines

loss) compared to PCC Vivace. Though one might argue that high congestion loss is fine as long as the final goodput reaches full link utilization, this is often not true, e.g., high loss rate can cause additional delays for key frames, resulting in lagging-experience in interactive or video streaming applications; or the large amount of transmission can cause energy burden on mobile devices.

In light of this important discovery, we urge future congestion control designs to carefully consider this tradeoff. Though PCC Allegro also achieves 5% random loss resilience similar to PCC Vivace, due to its naïve rate control algorithm, it pays a much higher convergence loss (9%) price. We also observe that though Vivace-Latency maintains low loss rate when there is only a single flow, its loss rate still grows as the concurrent sender number increases. BBR, positioned as a latency-aware protocol, also has the same effect and in fact, worse than Vivace-Latency. For Vivace-Latency, we believe it is because of increased noise of performance measurements due to dynamic interaction between a large number of concurrent flows, and aim to improve that in future works.

4.5.3 Improved Friendliness to TCP

To understand the TCP friendliness properties of various newly proposed protocols, we set up a 30ms RTT bottleneck link with one flow using the new protocol (BBR, PCC Allegro, or PCC Vivace) to compete with increasing number of CUBIC flows. As the number of senders increases, we also increase the total bandwidth and bottleneck buffer to maintain the same *per-flow share*. We used two per-flow share settings: (4Mbps, 30KB) and (8Mbps, 5KB), corresponding to 2BDP and 0.12BDP of buffer size, respectively. Figure 4.13 shows the ratio between the throughput of the flow using various new protocols and average throughput per CUBIC flow. A ratio of 1 in Figure 4.13 indicates perfect friendliness.

Vivace-Latency behaves as expected in the design (§4.3.4). When the number of CUBIC flows is small, since the queue is not always full and Vivace-Latency can reduce the rate to decrease the queue size, it achieves lower throughput than CUBIC flows. As the number of CUBIC senders increases, Vivace-Latency's micro-temporal behavior matches our expectation and achieves the best fairness among new generation protocols when number of CUBIC sender increases. Would Vivace-Latency on the Internet still be conservative when the number of competing CUBIC flows is small? Only large scale deployment experiences can tell for sure, but our real world experiments in § 4.5.5 strongly suggest that even using PCC Vivace in this conservative way, one can still achieve significantly higher performance compared to CUBIC, since CUBIC is simply not efficiently utilizing all available network resources. In addition, we can tune the latency penalty parameter b as we gain more real-world experiences.

BBR yields the worst TCP friendliness among protocols evaluated. In [3] and [28], BBR's TCP friendliness is claimed to be satisfied and evaluated in a setting where a single BBR flow competes with a single CUBIC flow when the buffer is large (2 BDP). We successfully reproduced this specific result (see 4Mbps/30KB(2BDP), BBR line) in Figure 4.13 when number of CUBIC flows is one. However, we discovered that as we add more CUBIC flows, BBR becomes increasingly aggressive: it effectively treats all competing CUBIC flows as a single "bundle" with its throughput ratio increasing linearly before it converges (with 16 CUBIC flows). Therefore, in practice, BBR can be very unfriendly to CUBIC when there are multiple competing CUBIC flows present.

Though PCC Allegro and Vivace-Loss both use loss-based utility functions, PCC Vivace is

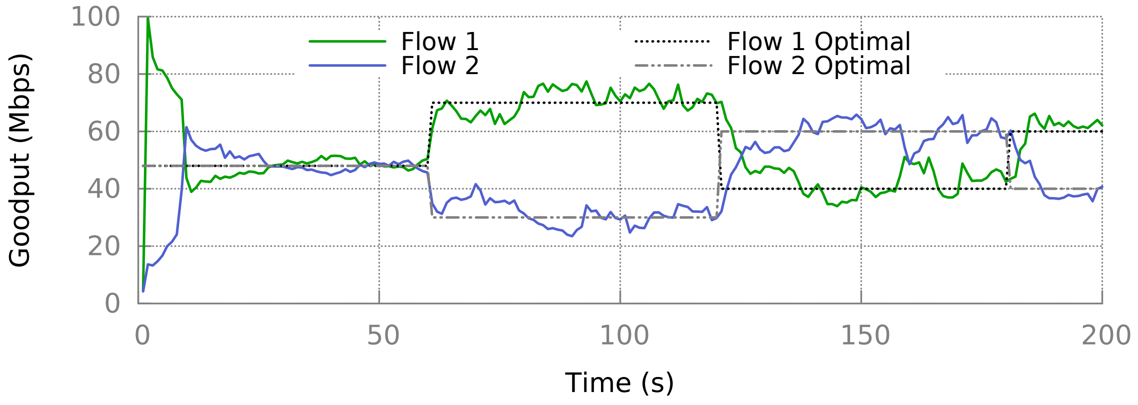


Figure 4.14: Flexible equilibrium by tuning utility knobs

much more agile in its reaction to competing TCP flows, especially under a shallow buffered network: its throughput ratio converges at 2.5 vs. PCC Allegro's 8 with shallow buffer. In addition, though Vivace-Loss dominates CUBIC when concurrent number of flow is small, the final converged throughput ratio (about 5 at 2BDP) is not nearly as bad as BBR and we believe it also qualifies as a potential deployment candidate. In sum, though perfect TCP friendliness is fundamentally hard, we believe that PCC Vivace provides a viable way towards adoption.

4.5.4 Flexible Convergence Equilibrium

With its unique utility function framework (§ 4.3.6), PCC Vivace unleashes the potential to be flexible and centrally controlled. To demonstrate this capability experimentally, we setup a 100Mbps, 30ms RTT link with two competing flows. As shown in Figure 4.14, we control the two flows' bandwidth share by changing their utility functions at 60s, 120s and 180s. The actual sending rate closely tracks the ideal allocation (dashed lines). This is only to show the prospect of this unique capability of PCC Vivace, and we will discuss a full fledged system leveraging this capability in future work.

4.5.5 Benefits in the Real-World

Finally, we evaluate PCC Vivace's real-world performance in the wild Internet. We set up senders at 3 different residential WiFi networks and receivers at 14 Amazon Web Service (AWS) sites, *i.e.*,

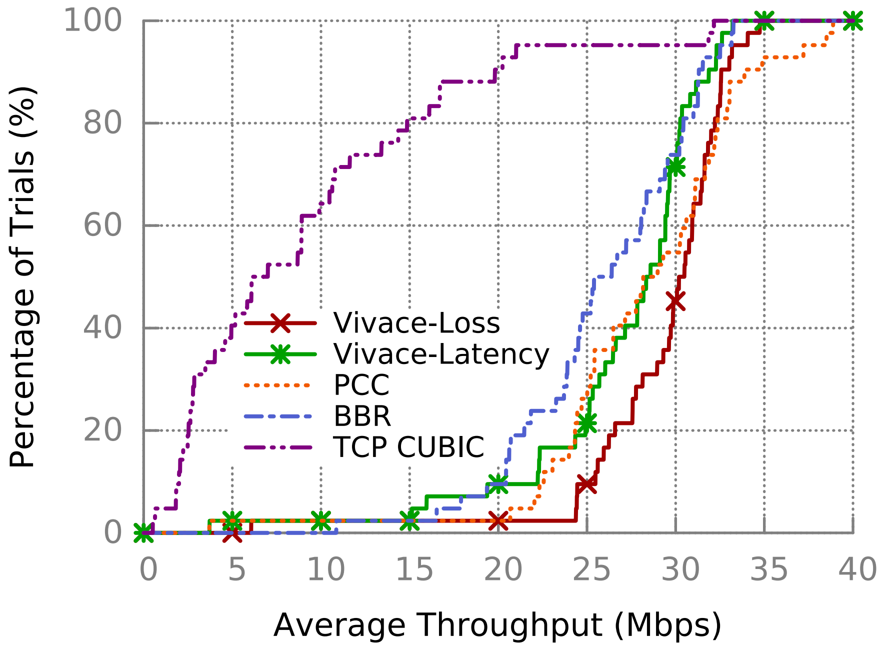


Figure 4.15: Performance gain in wild Internet

42 sender-receiver pairs.[4] As WiFi networks have more noise in latency than wired networks, we use a slightly larger $flt_{latency} = 0.05$ to filter small variation of latency.[5] For each AWS site we test all protocols, and compute the average throughput of each protocol from five 100s transmissions. Figure 4.15 shows the cumulative distribution of average throughput. Similar to the results in controlled networks, Vivace-Loss is slightly better than Vivace-Latency, and they both outperform BBR and CUBIC. Specifically, Vivace-Loss has a median throughput gain of 7.2%, 18.9%, and 4.0× compared with PCC Allegro, BBR, and CUBIC, respectively.

More importantly, even though having the possibility to be overly friendly to TCP, Vivace-Latency successfully achieves 11.6% and 3.7× better throughput than BBR and CUBIC in median case, *i.e.*, CUBIC flows just cannot efficiently utilize available bandwidth. This serves as a strong validation that PCC Vivace provides a viable deployment path forward. Larger scale evaluation will be future work. Furthermore, corresponding to the results in § 4.5.3, we notice that the degradation of Vivace-Latency often happens when there is a sudden peak in latency, which may be caused by

[4]We do not test the downlink because the virtualized AWS server has imperfect pacing, which impacts the performance of both PCC Vivace and PCC Allegro, and to some extent, BBR.

[5]We expect that in a full implementation, PCC Vivace can automatically adjust $flt_{latency}$ when observing high latency variance.

temporary congestion on the path, or concurrent uplink transmission at the sender side associated to the same router.

At last, the difference between Vivace-Loss and PCC Allegro is because PCC Allegro's slow reaction to network condition variation, *e.g.* packet loss due to available bandwidth reduction or congestion. As a result, it may gain higher throughput occasionally (only after 90th percentile), but it will suffer from a higher packet loss rate and often yield more aggressive behavior comparing to PCC Vivace. At the same time, we find that BBR is more easily impacted on oversea connections. For example, in one test through Mumbai, Vivace-Loss/-Latency achieve 79.3%/70.6% higher throughput than BBR.

4.6 Related Work

This paper extensively discusses traditional TCP variants, Remy, PCC and BBR, and compares them with PCC Allegro. To put PCC Allegro in context with the large body of literature on congestion control, we now discuss other closely related work.

In-network feedbacks. One class of protocols improve congestion control by providing explicit in-network feedback (e.g. available bandwidth and ECN) for better informed decisions [21, 26, 52]. These protocols yield good performance, but have been proven to be hard to deploy: they require coordinated change of protocol or network devices. PCC Allegro on the other hand is compatible with the TCP message format, and only requires deployment at the sender, and is therefore readily deployable.

Specially engineered congestion control. Another class of the recent works targets TCP's poor performance in specific network scenarios like LTE networks [78, 80] or data center networks [21, 58, 79] by leveraging unique insights of particular networks' behavior models, special tools or explicit feedback from the receiver. Some of these works are also moving away from TCP's hardwired control mechanism and are more like BBR; e.g., Timely [58] uses the RTT-gradient as a control signal similar to PCC Allegro, but it still uses a hardwired rate control algorithm that maps events to fixed reactions. Protocols in this class provide significant performance gains, but only target very specific environments. Some of them also require changes at both endpoints [78], which may challenge deployment in practice.

Short flows. Some congestion control protocols aim to optimize performance of very short flows [55, 59]. These protocols are complementary to PCC Allegro, because short-flow optimization in many cases is an "open loop" problem (*i.e.* transfer as much data as possible in first few RTTs with very limited feedbacks) whereas PCC Allegro targets the "closed loop" phase of data transfer (*i.e.* meaningful feedbacks can be gathered with long enough data transfer). In fact PCC Allegro can utilize [55, 59] as its starting phase (replacing slow-start).

Data-Driven Networking. A very recent work suggests to investigate the grand challenge of how using data and statistics gathered from the network can help networking designs [51]. As PCC Allegro is data-driven, it concretely shows a way to handle some of the challenges presented in [51], such as reliability of performance statistics, and how to effectively leverage gathered data to optimize performance.

4.7 Conclusion

In this chapter, we propose PCC Vivace, a congestion control architecture based on machine learning theory. PCC Vivace leverages a novel latency-aware utility function framework with gradient-ascent-based online learning rate control to achieve provably fast convergence and fairness guarantees. Extensive experimentation reveals that PCC Vivace significantly improves upon the existing state of the art in terms of performance, convergence speed, reactiveness, TCP friendliness, and more. We leave the open research questions regarding centralized resource allocation via PCC Vivace's simple interface, further improvement of performance statistics via cross-flow data analytics, as suggested in [51], to the future. With all these extremely promising performance results and flexibility potentials, we next dive into our efforts to push PCC congestion control family to production and real world.

Chapter 5

Pushing PCC to Real-World Deployments

5.1 Introduction

PCC as an innovative congestion control architecture carries huge potential benefits of improving transport layer performance in Internet. We aim to maximize the real-world impact with our academic research. It is not an easy path to take: to actually harvest those benefits, understand real-world limitations and further improve PCC, we need to push PCC to carry real large-scale traffic for either actual end host users or enterprises. To reach that goal, we need to make PCC more easily accessible through building reliable and new software components that can be utilized in different transport performance optimization use cases. In this thesis, we explore the following two avenues towards that goal.

- The first option is to build an user-space TCP tunneling proxy that transparently optimize the performance of tunneled traffic. This will make PCC immediately available for use cases such as Wide Area Network acceleration and optimization or large file transfers used in scientific data transfer scenarios and in film production industry.
- The second option is to integrate with a widely used user-space transport control framework from Google, called QUIC [15]. As QUIC is serving traffic from all Chrome browsers to any Google services, this integration will make it possible for Google to deploy and use PCC on a large scale. More importantly, the large scale performance telemetry provided by QUIC framework can be used to further perfect PCC's rate control algorithm. So we deem that work also valuable and essential.

It is also entirely possible to build a sender-side kernel module for PCC. However, because the entire TCP architecture in kernel is built around the old TCP architecture, there are many

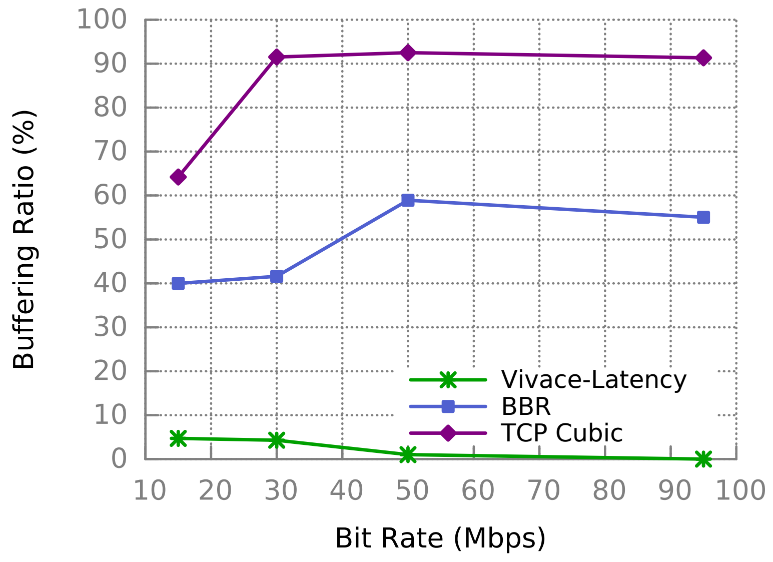


Figure 5.1: Video streaming buffering ratio with varying latency

missing constructs in the kernel for PCC to be functional. For example, how to implement monitor within the kernel TCP stack and how to control the sending rate precisely are all open questions. Therefore, we leave a full kernel implementation of PCC as future work.

5.2 PCC Tunneling Proxy

We implement a PCC tunneling proxy based on our user-space PCC implementations. When a client using PCC proxy tries to initiate a TCP connection to a PCC-enabled server, the client-side PCC daemon recognizes the connection and terminates that connection locally by pretending to be the server. At the same time, client-side PCC daemon initiates a PCC connection to the server-side PCC daemon. Server-side PCC daemon, up on establishing the connection, relays the connection to server applications (e.g. web servers) using TCP. After this process, a PCC tunnel is established. Server can transfer all its data to its local PCC daemon with very high performance, because the TCP connection is local. The same goes for client. In the middle, PCC carries data through the wild Internet. In this deployment mode, all existing clients' and servers' software stacks do not need to change besides installing PCC proxy in addition. This deployment mode can be used in software defined WAN (wide area network) solutions and CDN dynamic content acceleration solutions.

To understand how much application-level benefits this proxy can bring, we demonstrate using

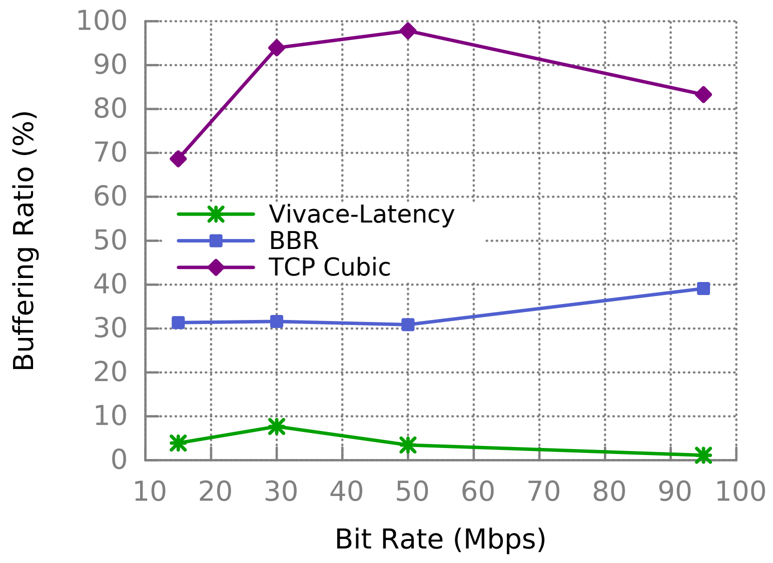


Figure 5.2: Video streaming buffering ratio with multiflow streaming

PCC Vivace proxy to tunnel RSTP-over-TCP between a pair of video streaming server and client. We compare PCC Vivace proxy's performance with BBR and TCP CUBIC using the metrics of streamed video's buffering ratio [31]. Buffering ratio is a commonly used performance metrics for video streaming smoothness, calculated as the ratio of time spent during buffering relative to the total streaming session time. We test with four 4K videos, with 15Mbps, 30Mbps, 50Mbps, and 95Mbps average bit rate requirements.

We first evaluate the buffering ratio with changing RTT between 10ms to 100ms in every five seconds. We set up a link with 300KB buffer and 0.01% random loss rate, with the network bandwidth of at least 10% more than the required bit rate to stream different videos. As shown in Figure 5.1, the average buffering ratio of PCC Allegro-Latency stays below 5%, corresponding to a reduction of at least 90% compared with both BBR and CUBIC.

To demonstrate the application-level benefit of PCC Allegro's stable convergence, we set up three competing streaming flows from three pairs of servers and clients. They share a bottleneck link with 75KB buffer, 100ms RTT, 0.01% random loss, and adequate bandwidth for all three to stream. As shown in Figure 5.2, PCC Allegro-Latency outperforms BBR and CUBIC by at least 76% and 90%, respectively. We attribute the degraded performance of BBR to its high throughput variance among flows.

We leave more evaluation of application level benefits, such as a cloud-deployed tunneling proxy for all Internet traffic for endusers as future research direction.

5.3 Integration with QUIC

QUIC (Quick UDP Internet Connections) is developed and used by Google as an higher performance and more flexible user-space alternative for kernel TCP stack. QUIC, as a full user-space transport layer software stack, has many exciting features such as improved multiplexing (i.e. multiples QUIC streams using a single UDP session), zero-RTT connection setups, dynamic connection migration, native integration with forward error correction mechanisms and etc. QUIC is deployed to support Google's applications without changing the kernel infrastructure. QUIC can also safely fall back to a TCP connection if end users are not using QUIC enabled Chrome browsers or UDP traffic is not allowed.

Among all the features, the most important feature in our view is QUIC's extensible congestion control framework. QUIC provides a elegantly architecture of congestion control interface that provides many nice functionality such as pacing based rate control interface and congestion event feedback interface. However, QUIC, out of box, still uses CUBIC (an improved version) as its congestion control algorithm. We believe integrating PCC with QUIC is very important because QUIC provides both a friendly environment to port PCC and a path to high impact and large-scale deployment of PCC.

As a first step, we integrated PCC Allegro with QUIC by implementing PCC Allegro on top of QUIC's congestion control interface. We overcome various engineering challenges including implementation of monitoring process on top of QUIC's infrastructure, handling unreliable or delayed measurement from QUIC's framework and accuracy of packet pacing. As of now, we have fully implemented PCC Allegro as a pluggable and functional congestion control module in QUIC.

Taking this implementation, we test its performance by deploying PCC Allegro in *production Google servers* and also by using a toy-server and a toy-client on Emulab. We choose to reproduce PCC Allegro's experiments for the lossy network condition and also for the mult-flow convergence experiment. We choose lossy network condition because it is a particularly challenging network condition and can demonstrate PCC's consistent high performance. We choose multi-flow convergence

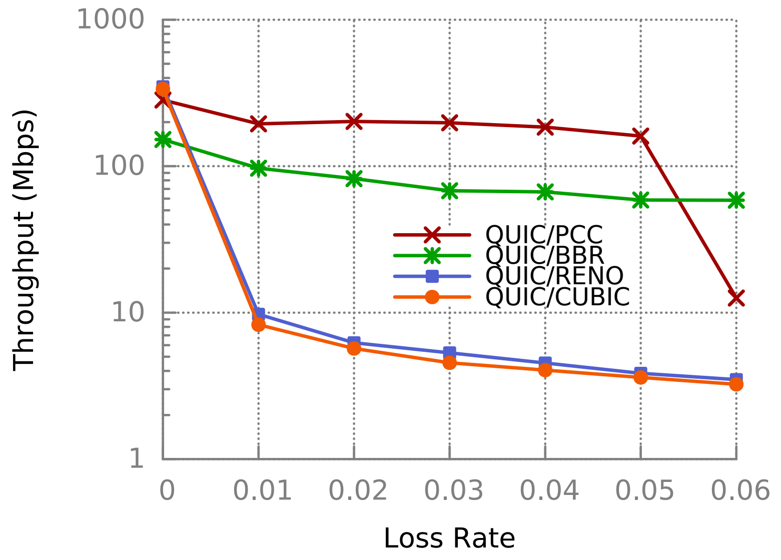


Figure 5.3: QUIC implementation replicates performance of user-space implementation

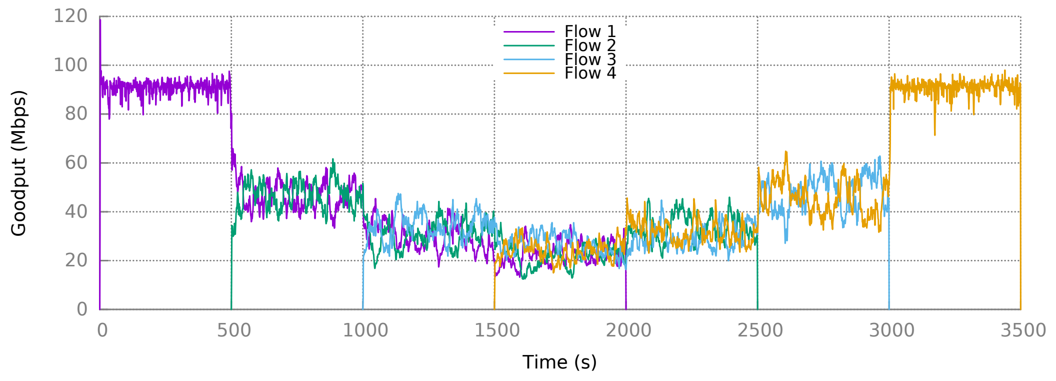


Figure 5.4: QUIC implementation replicates convergence of user-space implementation

because it demonstrates PCC's game-theory-driven dynamics.

Figure 5.3 shows the throughput comparison between different protocols implemented on top of QUIC under lossy network condition. In this experiment, we deploy PCC Allegro on QUIC in one Google Front End (GFE) server in Georgia and point a Chrome browser, running on a ethernet-connected server in our lab in UIUC, to this server. To conduct this experiment, we artificially inject different loss rates on the client side and we compare the average file downloading throughput for Google Drive over 100s. We can see that PCC Allegro on QUIC achieves very good feature parity with our user-space implementation, and achieves better throughput than CUBIC, RENO and BBR on top of QUIC.

Figure 5.4 shows the convergence behavior of PCC Allegro on QUIC. We conduct this experiment on Emulab using the same setting as described in § 3.4. The QUIC implementation also achieve very good feature parity with our user-space implementation. Both experiments show that our portation of PCC to QUIC is successful. We also preliminarily measured the throughput performance in a residential WiFi Internet condition using production deployment of PCC Allegro. We observe PCC-on-QUIC shows 6$\times$ throughput gain over CUBIC-on-QUIC.

Chapter 6

Conclusion

In this thesis, we propose Performance-oriented Congestion Control (PCC), a new congestion control architecture that achieves consistent high performance across a wide spectrum of challenging network conditions. First, we identify the fundamental flaw of traditional congestion control architecture and establish the foundation of PCC's new architecture. We use theoretical analysis to prove PCC's two-parted architecture: a flexible utility function framework and a learning rate control algorithm, is viable. Second, taking the theoretical insights and basic architecture, we implement the first protocol in PCC's family, PCC Allegro. Through extensive evaluations across a wide spectrum of challenging network conditions, we demonstrate PCC's often 10X performance improvements comparing to specifically engineered TCP variants. Third, we take a step back and review the design of PCC's two architectural components. Insipred by the principled methodologies of online learning theory, we propose PCC Vivace, the second protocol in PCC family with a novel latency-aware and strictly socially concave utility framework and a gradient-ascent-based online learning rate control algorithm. We overcome significant engineering and practical challenges to transform the theory insights to an efficient and robust implementation of PCC Vivace. Extensive experimentation reveals that PCC Vivace significantly improves over PCC Allegro and other state-of-art proposals, such as BBR, in terms of performance, convergence speed, reactiveness, TCP friendliness, and more. PCC Vivace also exposes a centrally controllable interface to dynamically and predictably control convergence point of multiple competing flows. In the process of designing PCC Vivace, we also discover a fundamental tradeoff hypothesis between random loss resilience and convergence state's congestion loss. Finally, we put significant efforts in pushing PCC to production by implementing a user-space two-sided tunneling proxy and integrating with Google's QUIC framework. Using the user-space proxy, we demonstrate that PCC can immediately bring benefits to different applications. We also successfully integrated and deployed PCC Allegro in

Google's production infrastructure. Real-world measurement results, though preliminary, suggest the portation retains important performance advantages of PCC.

With the rising tide of big data processing, Virtual Reality, hyper-scale geo-distributed data centers joining the already non-stopping explosion Internet applications, a consistent high performance transport layer is needed more than ever. Therefore, PCC's story is far from ending. There are multiple paths towards exciting future works. Roughly, they can be put into three categories as follows.

First is on keeping advancing PCC's technical advantages over existing and state-of-art protocols. In that front, we have already identified significant challenges to be solved such as an enhanced performance measurement quality-control layer to cleanse the noise in performance measurement results. For example, one possibility is to draw from the insights in the recent advance on deep learning to build a noise filter. Another example is to study how to fully utilize PCC's unique capability of tunable convergence point of competing flows. Yet another example is to answer the open question of whether in-network control or feedbacks, maybe using recent advance of programmable switches, still can help in the new architecture of PCC. If yes, to what extend?

Second is on pushing PCC to large scale deployment. We are going to keep pursuing the path of integration with QUIC by collaborating with Google. Hopefully, with large scale measurements, we can gain significant insights in PCC in the wild Internet and further perfect the algorithm and architecture.

Finally, we believe the research topic of congestion control in Internet is, though old, still extremely relevant and important as the scalability of Internet still very much depends on that. Therefore, we believe it is extremely important to raise the awareness of the importance of this issue and form vivid communities across academia and industry. We believe it is possible to form a consortium of congestion control with device vendors, content providers, service providers, PCC research team and other research teams to approach this important issue together.

References

[1] http://goo.gl/lGvnis.

[2] http://www.dataexpedition.com/.

[3] BBR talk in IETF 97. www.ietf.org/proceedings/97/slides/slides-97-iccrg-bbr-congestion-control-01.pdf.

[4] Browsers usually opens parallel TCP connections to boost performance. http://goo.gl/LT0HbQ.

[5] Cisco Forecast: 3.4 Devices Connected to the Internet Per Person by 2020. goo.gl/xmnoPw.

[6] Codel Linux implementation. http://goo.gl/06VQqG.

[7] ESNet. http://www.es.net/.

[8] GENI testbed. http://www.geni.net/.

[9] Internet of Things (IoT): number of connected devices worldwide from 2012 to 2020 (in billions). goo.gl/C0X1v3.

[10] Internet2 ION service. http://webdev0.internet2.edu/ion/.

[11] Level 3 Bandwidth Optimizer. http://goo.gl/KFQ3aS.

[12] Limelight Orchestrate(TM) Content Delivery. http://goo.gl/M5oHnV.

[13] List of customers of a commercial high speed data transfer service provider. http://goo.gl/kgecRX.

[14] PCP user-space implementation. http://homes.cs.washington.edu/~arvind/pcp/pcp-ulevel.tar.gz.

[15] QUIC, a multiplexed stream transport over UDP. www.chromium.org/quic.

[16] Report mentioning revenue range of one high speed data transfer service company. http://goo.gl/7mzB0V.

[17] Satellite link latency. http://goo.gl/xnqCih.

[18] TelliShape per-user traffic shaper. http://tinyurl.com/pm6hqrh.

[19] UDT: UDP-based data transfer. udt.sourceforge.net.

[20] PlanetLab — An open platform for developing, deploying, and accessing planetary-scale services. July 2010. http://www.planet-lab.org.

[21] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP. *Proc. of ACM SIGCOMM*, September 2010.

[22] T. Alpcan and T. Basar. A utility-based congestion control scheme for Internet-style networks with delay. *Proc. IEEE INFOCOM*, April 2003.

[23] T. Anderson, A. Collins, A. Krishnamurthy, and J. Zahorjan. PCP: Efficient endpoint congestion control. *Proc. of NSDI*, May 2006.

[24] L. Brakmo, S. Lawrence, S. O'Malley, and L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. *Proc. of ACM SIGCOMM*, 1994.

[25] H. Bullot and R. L. Cottrell. Evaluation of advanced tcp stacks on fast longdistance production networks. *Proc. PFLDNeT*, February 2004.

[26] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and K. van der Merwe. Design and implementation of a routing control platform. *Proc. of NSDI*, April 2005.

[27] C. Caini and R. Firrincieli. TCP Hybla: a TCP enhancement for heterogeneous networks. *International Journal of Satellite Communications and Networking*, August 2004.

[28] N. Cardwell, Y. Cheng, C. Gunn, S. Yeganeh, and V. Jacobson. BBR: Congestion-based congestion control. *Queue*, 14(5):50, 2016.

[29] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding TCP incast throughput collapse in datacenter networks. *Proc. ACM SIGCOMM Workshop on Research on Enterprise Networking*, August 2009.

[30] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. August 2014.

[31] 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. *Proc. of ACM SIGCOMM*, August 2011.

[32] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. PCC: Re-architecting congestion control for consistent high performance. *Proc. of NSDI*, March 2015.

[33] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control and why this means we need new algorithms. *ACM Computer Communication Review*, January 2006.

[34] N. Dukkipati, N. McKeown, and A. Fraser. RCP-AC: congestion control to make flows complete quickly in any environment. *Proc. IEEE INFOCOM*, April 2006.

[35] ESnet. Virtual Circuits (OSCARS), May 2013. http://goo.gl/qKVOnS.

[36] E. Even-Dar, Y. Mansour, and U. Nadav. On the convergence of regret minimization dynamics in concave games. 2009.

[37] M. E. Even-Dar, Y. Mansour, and U. Nadav. On the convergence of regret minimization dynamics in concave games. *Proc. of ACM symposium on Theory of computing*, 2009.

[38] Y. Ganjali and N. McKeown. Update on buffer sizing in internet routers. *ACM Computer Communication Review*, October 2006.

[39] J. Gettys and K. Nichols. Bufferbloat: Dark buffers in the Internet. *ACM Queue*, December 2011.

[40] M. Ghobadi, S. Yeganeh, and Y. Ganjali. Rethinking end-to-end congestion control in software-defined networks. *Proc. of HotNets*, November 2012.

[41] Y. Gu, X. Hong, M. Mazzucco, and R. Grossman. SABUL: A high performance data transfer protocol. *IEEE Communications Letters*, 2003.

[42] S. Gutz, A. Story, C. Schlesinger, and N. Foster. Splendid isolation: a slice abstraction for software-defined networks. *Proc. ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, August 2012.

[43] S. Ha, I. Rhee, and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review*, 2008.

[44] E. Hazan. Introduction to online convex optimization. http://ocobook.cs.princeton.edu/OCObook.pdf.

[45] C. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. *Proc. of ACM SIGCOMM*, August 2013.

[46] Y. Hu and V. Li. Satellite-based Internet: a tutorial. *Communications Magazine*, 2001.

[47] V. Jacobson. Congestion avoidance and control. In *ACM Computer Communication Review*, 1988.

[48] M. Jain and C. Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. *Proc. Passive and Active Measurement (PAM)*, March 2002.

[49] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, and M. Zhu. B4: Experience with a globally-deployed software defined wan. *ACM Computer Communication Review*, September 2013.

[50] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *Proc. of CoNEXT*, December 2012.

[51] J. Jiang, S. Sun, V. Sekar, and H. Zhang. Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation. *Proc. of NSDI*, March 2017.

[52] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. *Proc. of ACM SIGCOMM*, August 2002.

[53] F. Kelly, A. Maulloo, and D. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society*, 1998.

[54] S. Keshav. *The packet pair flow control protocol.* ICSI, 1991.

[55] Q. Li, M. Dong, and P. Godfrey. Halfback: Running short flows quickly and safely. *Proc. of CoNEXT*, November 2015.

[56] S. Liu, T. Başar, and R. Srikant. TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks. *Performance Evaluation*, 2008.

[57] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang. TCP Westwood: bandwidth estimation for enhanced transport over wireless links. *Proc. ACM Mobicom*, July 2001.

[58] R. Mittal, N.Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats. TIMELY: RTT-based congestion control for the datacenter. *ACM Computer Communication Review*, August 2015.

[59] R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker. Recursively cautious congestion control. *Proc. of NSDI*, March 2014.

[60] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking (ToN)*, 2000.

[61] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan. Mahimahi: Accurate record-and-replay for HTTP. *Proc. USENIX ATC*, August 2015.

[62] K. Nichols and V. Jacobson. Controlling queue delay. *Communications of the ACM*, 2012.

[63] H. Obata, K. Tamehiro, and K. Ishida. Experimental evaluation of TCP-STAR for satellite Internet over WINDS. *Proc. Autonomous Decentralized Systems (ISADS)*, June 2011.

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

[65] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos. Elastic-Switch: practical work-conserving bandwidth guarantees for cloud computing. *Proc. of ACM SIGCOMM*, August 2013.

[66] R. Prasad, C. Dovrolis, and M. Thottan. Router buffer sizing revisited: the role of the output/input capacity ratio. *Proc. of CoNEXT*, December 2007.

[67] B. Raghavan and A. Snoeren. Decongestion control. *Proc. of HotNets*, 2006.

[68] J. Rosen. Existence and uniqueness of equilibrium point for concave n-person games. *Econometrica*, 1965.

[69] A. Sivaraman, K. Winstein, S. Subramanian, and H. Balakrishnan. No silver bullet: extending SDN to the data plane. *Proc. of HotNets*, July 2013.

[70] A. Sivaraman, K. Winstein, P. Thaker, and H. Balakrishnan. An experimental study of the learnability of congestion control. *Proc. of ACM SIGCOMM*, August 2014.

[71] L. Tan, C. Yuan, and M. Zukerman. FAST TCP: Fairness and queuing issues. *IEEE Communication Letter*, August 2005.

[72] VSAT Systems. TCP/IP protocol and other applications over satellite. http://goo.gl/E6q6Yf.

[73] G. Vu-Brugier, R. Stanojevic, D. J. Leith, and R. Shorten. A critique of recently proposed buffer-sizing strategies. *ACM Computer Communication Review*, 2007.

[74] D. Wei, C. Jin, S. Low, and S. Hegde. FAST TCP. *IEEE/ACM Trans. Networking*, December 2006.

[75] D. Wei, C. Jin, S. Low, and S. Hegde. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Transactions on Networking*, 2006.

[76] B. White, J. Lepreau, L. Stoller, R. Ricci, G. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. *Proc. of OSDI*, December 2002.

[77] K. Winstein and H. Balakrishnan. TCP ex Machina: computer-generated congestion control. *Proc. of ACM SIGCOMM*, August 2013.

[78] K. Winstein, A. Sivaraman, and H. Balakrishnan. Stochastic forecasts achieve high throughput and low delay over cellular networks. *Proc. of NSDI*, March 2013.

[79] H. Wu, Z. Feng, C. Guo, and Y. Zhang. ICTCP: Incast congestion control for TCP in data center networks. *Proc. of CoNEXT*, November 2010.

[80] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg. Adaptive congestion control for unpredictable cellular networks. *Proc. of ACM SIGCOMM*, August 2015.

[81] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, pages 928–936. AAAI Press, 2003.

[82] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. 2003.