PCC Vivace: Online-Learning Congestion Control

Mo Dong and Tong Meng, *UIUC;* Doron Zarchy, *The Hebrew University of Jerusalem;*
Engin Arslan, *UIUC;* Yossi Gilad, *MIT;* Brighten Godfrey, *UIUC;*
Michael Schapira, *The Hebrew University of Jerusalem*

https://www.usenix.org/conference/nsdi18/presentation/dong

This paper is included in the Proceedings of the
15th USENIX Symposium on Networked
Systems Design and Implementation (NSDI '18).

April 9–11, 2018 • Renton, WA, USA

ISBN 978-1-939133-01-4

Open access to the Proceedings of
the 15th USENIX Symposium on Networked
Systems Design and Implementation
is sponsored by USENIX.

PCC Vivace: Online-Learning Congestion Control

Mo Dong*, Tong Meng*, Doron Zarchy†, Engin Arslan‡, Yossi Gilad§,
P. Brighten Godfrey* and Michael Schapira†

*UIUC, †Hebrew University of Jerusalem, ‡University of Nevada, Reno, §MIT

Abstract

TCP's congestion control architecture suffers from notoriously bad performance. Consequently, recent years have witnessed a surge of interest in both academia and industry in novel approaches to congestion control. We show, however, that past approaches fall short of attaining ideal performance. We leverage ideas from the rich literature on *online (convex) optimization* in machine learning to design Vivace, a novel rate-control protocol, designed within the recently proposed PCC framework. Our theoretical and experimental analyses establish that Vivace significantly outperforms traditional TCP variants, the previous realization of the PCC framework, and BBR in terms of performance (throughput, latency, loss), convergence speed, alleviating bufferbloat, reactivity to changing network conditions, and friendliness towards legacy TCP in a range of scenarios. Vivace requires only sender-side changes and is thus readily deployable.

1 Introduction

The recent surge of interest in both academia and industry in improving Internet congestion control [8, 11, 13, 19, 21, 22, 24, 25, 31, 32, 36] has made it apparent that today's prevalent congestion control algorithms, the TCP family, fall short of important performance requirements. Indeed, transport rate control faces numerous challenges. First and foremost, a congestion control architecture should be able to efficiently utilize network resources under varying and complex network conditions. This includes optimizing for throughput, loss, and latency, and doing so in a plethora of environments — potentially with non-congestion loss [8], high-RTT cross-continent links, highly dynamic networks such as WiFi and LTE links, etc. Second, congestion control should guarantee quick convergence to stable and fair rates 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.

Traditional algorithms [6, 15, 23] fail to satisfy the first two requirements; their performance can be as high as 10× away from the optimal under non-congestion packet loss [11]. Recent proposals, including Remy [31], PCC [11], and BBR [8], investigate new approaches to this challenge. Remy replaces the human designer with an offline optimization scheme that searches for the best scheme within a certain design space, for a pre-specified range of network conditions. While they can attain high performance, Remy-generated TCPs are inherently prone to degraded performance when the actual network conditions deviate from input assumptions [27].

BBR takes a white-box network-modeling approach, translating change patterns in performance measurements (e.g., increase in delivery rate) to presumed underlying network conditions (e.g., bottleneck throughput and latency). PCC takes a black box approach: a PCC sender observes performance metrics resulting from sending at a specific rate, converts these metrics into a numerical utility value, and adapts the sending rate in the direction that empirically is associated with higher utility. Our experiments indicate that while improving substantially over traditional schemes, both the *specific* realization of PCC in [11], termed "PCC Allegro" (or simply Allegro) henceforth, and BBR's implementation [8], 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 overly long. In addition, when BBR's model of the network does not reflect the complexities of reality, performance can suffer severely. Lastly, they are both highly aggressive towards TCP, although BBR is designed with TCP-friendliness in mind.

To address the above limitations, we draw inspiration from literature on online (convex) optimization [12, 16, 37] to design PCC Vivace, a novel congestion control

scheme. Vivace adopts the high-level architecture of PCC – a utility function framework and a learning rate-control algorithm – but realizes both components differently. First, Vivace relies on a new, learning-theory-informed framework for utility derivation that incorporates crucial considerations such as latency minimization and TCP friendliness. Second, Vivace employs provably (asymptotically) optimal online optimization based on gradient ascent to achieve high utilization of network capacity, swift reaction to changes, and fast and stable convergence. In particular, our contributions are:

(1) A principled framework for transport utility with multiple novel consequences. We prove that for a proper choice of utility functions that incorporate not only throughput and loss (as in Allegro), but also latency, a stable global rate configuration (Nash equilibrium) always exists; show a tradeoff between random loss tolerance and packet loss at convergence with competing senders; and allow controllable capacity allocation among competing senders with heterogeneous utilities (suggesting a future opportunity for centralized network control in an SDN or OpenTCP [13] architecture).

(2) A rate control scheme that utilizes gradient-ascent algorithms from online learning theory to achieve an improved tradeoff between stability and reactivity. We prove that our rate-control scheme guarantees quick convergence to the equilibrium guaranteed by our choice of utility functions, and employ additional techniques to improve rate control in the face of noisy measurements, such as linear regression and low-pass filtering.

(3) Extensive experiments with PCC Vivace, PCC Allegro, BBR, and various TCP variants, in controlled environments, real residential Internet scenarios, and with video-streaming applications. Highlights include: improved performance in rapidly changing conditions (70% less packet loss and 72.5% higher throughput than PCC Allegro, and around 20% median throughput gain over BBR); convergence about $2\times$ faster than Allegro and stability about $2\times$ better than BBR; 57% less video buffering time than BBR with multiple ongoing streams; and significantly improved TCP friendliness.

By no means do we expect that Vivace is the end of the story. Optimizing rate quickly and accurately with limited information in a complex, noisy environment is difficult, and we highlight a simulated LTE environment where a "white-box" model-based approach engineered for this context, namely Sprout [32], outperforms Vivace, as a case for future work. However, Vivace represents a substantial overall advance, showing how a strong learning-theoretic basis yields practical improvements.

2 Rate-Control Through Online Learning

When approached from an online learning perspective, the challenges outlined in § 1 fall naturally into the category of online optimization in machine learning and game theory [16, 37] (a.k.a. "no-regret learning"). Online learning provides a useful and powerful abstraction for decision making under uncertainty. In the online learning setting, a *decision maker* repeatedly selects between available *strategies*. Only after selection is the decision maker aware of the implications of the selected strategy, in terms of a resulting *utility* value. State-of-the-art online learning algorithms provide provable guarantees (namely, the classical "no regret" guarantee [16, 37]) even under *complete* uncertainty about the environment, i.e., without assuming/inferring *anything* about the relation between choices of strategies and the induced utility values. In addition, results in game theory establish that online learning algorithms "play well" together, in the sense that, under the appropriate conditions, global convergence to a stable equilibrium is guaranteed when there are multiple decision makers.

We are thus inspired to apply ideas and machinery from online learning to rate control on the Internet, which can naturally be cast as an online learning task as follows. A traffic sender repeatedly selects between sending rates. After sending at a certain rate for "sufficiently long", the sender learns its performance implications by translating aggregated statistics (e.g., achieved goodput, packet loss rate, average latency) into a numerical utility value, and then adapts the sending rate in response. Importantly, the application of online learning to rate-control is particularly challenging since often only very limited feedback from the network is available to the sender, and so accurately determining the utility derived from sending at a certain rate is sometimes infeasible.

PCC [11] is a promising step towards online-learning-based congestion control. The gist of its architecture is as follows. Time is divided into consecutive intervals, called Monitor Intervals (MIs), each devoted to "testing" the implications for performance of sending at a certain rate. PCC aggregates selective ACKs for packets sent in a MI into the above-mentioned meaningful performance metrics, and feeds these metrics into a utility function that translates them into a numerical value. PCC's rate-control module continuously adjusts the sending rate in the direction that is most beneficial in terms of utility.

However, the specific realization of PCC Allegro in [11] is far from tapping the full potential of online learning. First, Allegro uses a somewhat arbitrary choice of utility function. While [11] proves this choice induces desirable properties in some settings, fair convergence is not provably guaranteed when utility functions are latency-aware, reasoning about fundamental tradeoffs in parameter settings is difficult, and there is no theoretical understanding of what happens when Allegro senders with different utility functions interact with each other.

Second, Allegro inherently ignores the information re-

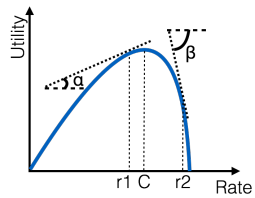


Figure 1: Utility-based rate-control

flected in the utility when deciding on step size. Suppose Allegro's utility function is as described in Figure 1, where C is the capacity of a single link. Then, consider two possible initial rates for the Allegro-sender: r_1 and r_2 ($r_1 < C \ll r_2$). When starting at r_1, the Allegro sender will increase its rate to $r_1(1+\varepsilon)$, for a fixed $\varepsilon > 0$, whereas the initial rate r_2 will be followed by a decrement to $r_2(1-\varepsilon)$. Intuitively, this rate-adjustment is not optimal; a small ε will result in slowly lowering the rate from r_2, leading to long convergence time, whereas choosing too large an 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 or convergence.

The combination of Allegro's fairly naive rate control scheme and its ad hoc choice of utility function prevents it from attaining good performance under rapidly changing network conditions, does not alleviate bufferbloat, results in convergence rate/stability tradeoff that is better TCP's yet still suboptimal, leads to high packet loss upon convergence, and is overly aggressive towards TCP. Hence, despite a promising architecture, the operational instantiation of PCC in [11] is still far from optimal.

To address the above limitations, Vivace's design borrows ideas from the rich body of literature on online convex optimization [12, 16, 37] to replace the realization of the two crucial components of PCC's high-level architecture: (1) the utility function framework, and (2) the learning rate-control algorithm. First, Vivace relies on a new, learning-theory-informed framework for utility derivation [12], which guarantees multiple competing Vivace senders will converge to a unique stable rate configuration that is fair and near-optimal. Second, Vivace employs provably optimal gradient-ascent-based no-regret online optimization [37] to adjust sending rates, taking into account not only the direction (increase/decrease) that is more beneficial utility-wise, but also the *extent* to which increasing/decreasing the rate impacts utility.

No-regret learning. The classical objective in online learning theory is *regret minimization*. We give an informal exposition of the implications of no-regret for congestion control here. See [16, 37] for a more complete treatment. A "no-regret" rate-control protocol, such as Vivace, guarantees that its choices of rates are asymptotically (across time) no worse, utility-wise, than sending at what would have been (in hindsight) the *best fixed* rate. No-regret is a useful guarantee for two reasons.

First, no-regret provides a formal performance guarantee for individual senders across *all* network conditions, even highly dynamic or *adversarially* chosen (within the scope of the model). We believe Vivace is the first congestion control scheme to provide such a guarantee.

Second, no-regret provides a powerful lens for theoretical analysis, which we will use to reason formally about convergence with multiple competing no-regret senders, even with heterogeneous utility functions across the senders, and also about tradeoffs between resilience to non-congestion loss and loss upon convergence.

Limitations of no-regret. As no-regret relates performance to the best *fixed* strategy, the quality of this guarantee in a dynamic environment depends on the speed at which the protocol minimizes "regret" [16]. If, from an arbitrary starting state, the regret vanishes to a desired low value within T time units, then the no-regret guarantee applies relative to the best fixed strategy within *every* T units of time. Empirically (§5.1.4), Vivace adapts quickly to changes in network conditions.

Of course, a guarantee of near-optimality relative to the best *dynamic* strategy would be even better. However, such guarantees often entail assumptions about the environment, e.g., that the network behavior exhibits high regularity. Vivace reflects the design choice of avoiding such assumptions. That said, an important direction for future research is to quantify to what extent real-world networks are sufficiently predictable (e.g., via machine learning) to improve rate selection.

PCC Allegro vs. PCC Vivace. Compared with PCC Allegro, PCC Vivace's utility framework (1) incorporates latency awareness, mitigating the bufferbloat problem and the resulting packet loss and latency inflation, (2) extends to heterogeneous senders with different utility functions, enabling flexible network-resource allocation, and (3) induces more friendly behavior towards TCP, and thus is better suited for real-world deployment. In addition, Vivace's rate-control algorithm (1) provides faster, more stable convergence, and (2) reacts more quickly upon changes to network conditions.

3 Vivace's Utility Framework

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

$$u\left(x_i, \frac{d(RTT_i)}{dT}, L_i\right) = x_i^t - bx_i\frac{d(RTT_i)}{dT} - cx_i \times L_i, \quad (1)$$

where $0 < t < 1, b \geq 0, c > 0$, 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. The parameters b, c, t are constants. Intuitively, utility functions of the above

form reward increase in throughput (via x_i^t), and penalize increase in both latency ($bx_i \frac{d(RTT_i)}{dT}$) and loss ($cx_i \times L_i$). We next identify the properties of utility functions within our framework, and refer to [2] for formal analysis.

To see why Vivace's utility function does not consider the absolute value of latency, instead using RTT gradient, consider the following example. A single sender on a link with a large buffer sends at a rate of twice the capacity of the link for a single MI; then, in the next MI, it tries a slightly lower but still over-capacity rate. Such a sender would experience higher absolute latency in the second MI than in the first MI (since the link's queue is only further lengthened), even though lowering the rate was clearly the right choice. To learn within a single MI that lowering the rate is more beneficial, the sender examines the rate at which latency increases or decreases.

The choice of values for the parameters b, c and t in the utility function have crucial implications for the existence of an equilibrium point when multiple Vivace senders compete, and for the latency and congestion loss in such an equilibrium. Due to space limitations, the full proofs of the theorems in this section appear in [2].

3.1 Stability and Fairness

When $t \leq 1$, the family of utility functions in Equation 1 falls into the category of "socially-concave" in game theory [12]. A utility function within this category, when coupled with a theoretical model of Vivace's online-learning rate-control scheme (described in § 4), guarantees high performance from the individual sender's perspective and ensures quick convergence to a global rate-configuration [16, 37]. Specifically, we consider a network model with n senders competing on a bottleneck link with a FIFO queue. The following theorem shows convergence to a fair equilibrium.

Theorem 1. *When n Vivace-senders share a bottleneck link, and each Vivace-sender i's utility function is defined as in Eq. 1, the senders' sending rates converge to a fixed configuration* $(x_1^*, \ldots, x_n^*)$ *such that* $x_1^* = x_2^* = \ldots = x_n^*$.

We further analyze the latency in equilibrium. Ideally, upon convergence the latency will not exceed the base RTT, i.e., the RTT when the link buffer is not occupied. Theorem 2 shows how Vivace can achieve that through a proper assignment of value for the parameter b.

Theorem 2. *Let C denote the capacity of the bottleneck link. If* $b \geq tn^{2-t}C^{t-1}$*, then the latency in the unique stable configuration is the base RTT.*

3.2 Random Loss *vs.* Congestion

Non-congestion packet loss (due to lossy wireless links, port flaps on routers, *etc.*) is a common phenomenon in today's Internet [8]. We say a rate-control protocol is *p-loss-resilient* if that protocol does not decrease its sending rate under random loss rate of at most p.

For Vivace to be p-loss-resilient, we need to set c in the above utility function framework to be $c = \frac{tC^{t-1}}{p}$ (details in [2]). However, **enduring more random loss comes at a price**. To simplify the analysis, assume that $b = 0$ (i.e., the utility function is purely loss-based). The following theorem captures a link between random loss resilience and loss due to many competing senders.

Theorem 3. *In a system of n Vivace senders, each p-loss-resilient, the loss rate L of each sender i in equilibrium (with no random loss) satisfies*

$$p = \frac{nL - L + 1}{(1-L)^{1-t} n^{2-t}}. \tag{2}$$

To illustrate Theorem 3 with simplified algebra,[1] suppose that $t = 1$ and $b = 0$. Enduring random loss rate of p implies that the derivative of the utility function u satisfies:

$$\dot{u} = 1 - cp \geq 0, \quad \text{and so:} \quad c \leq 1/p$$

By plugging $t = 1$ in Equation 2, we find that in a system of n Vivace senders sharing a link, the loss rate experienced by each sender under equilibrium (to which Vivace is guaranteed to converge, by Theorem 1), given $n > c$ (if $n < c$, then $L = 0$), is $L = p\frac{n-1/p}{n-1}$.

When $n \to \infty$, the congestion loss rate on convergence approaches the random loss resilience p! Therefore, **withstanding more random loss comes at the cost of suffering more loss upon convergence for a large number of senders**. Our experiments with TCP, BBR, and Allegro, show a similar tradeoff, indicating that this is a barrier for current congestion control frameworks.

3.3 Heterogeneous Senders

So far, our discussion focused on the environment where the senders are homogeneous, i.e., they employ the same utility function. However, our utility framework allows us to reason about interactions between heterogeneous Vivace senders competing over a shared link.

Recent studies on SDN-based traffic engineering [17, 18] and network optimization for big-data systems [9] 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 [13] 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.

[1] Our theorems require $t < 1$, but t can be arbitrarily close to 1.

Vivace's utility function framework, in contrast, provides flexibility in resource-allocation. As a concrete example, consider the following loss-based utility:

$$u(x_i, L_i) = x_i - c_i x_i \left(\frac{1}{1 - L_i} - 1 \right) \quad (3)$$

This utility function, similar to that of § 3, induces a unique stable rate configuration to which Vivace senders are guaranteed to converge (more formally, it is also "socially concave"). 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 $\sum_{j \in N} x_j = C$. Then we have:

Theorem 4. *A system of n Vivace senders, in which each sender's utility function is of the form in Equation 3, converge to rate configuration $x_1^*, x_2^*, \ldots, x_n^*$, if for each $i \in [n]$, the loss penalty coefficient in Equation 3 is set to $c_i = \frac{C}{x_i^*}$.*

Hence, one can flexibly adjust the bandwidth allocated to each sender at equilibrium by tuning Vivace's parameters $\{c_i\}$. We experimentally validate this result in § 5.

4 Vivace's Rate Control

Vivace's rate control begins with a **slow start** phase in which the sender doubles sending rate every MI and permanently exits slow start when its empirically-derived utility value decreases for the first time. It then enters the **online learning** phase, which we focus on here.

4.1 Key Idea and Challenges

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 [12, 16, 37]. Specifically, when the utility functions are strictly convex, which is satisfied when $t < 1$ in our utility function formulation (Equation 1), the following two desiderata are fulfilled (see [2] for proofs). (1) Each sender is guaranteed that employing Vivace is (asymptotically) no worse than the optimal *fixed* sending rate in hindsight, termed the "no-regret" guarantee in online learning literature [12, 16, 37]. This is a strong guarantee in that it applies even when the sender is presented with adversarial environments, but it is limited in the sense that it quantifies performance with respect to the actual history of experienced network conditions and not to the conditions that would have resulted from sending at other rates. (2) When multiple senders share the same link, quick convergence to an equilibrium point is guaranteed.

In theory, applying gradient ascent to rate-control means starting at some initial transmission rate and repeatedly estimating γ, by which we denote the gradient (with respect to sending rate) of the utility function, through sampling and changing the rate by $\theta\gamma$, where θ is initially set to be a very high positive number. With time, θ gradually diminishes to 0. But realizing this in practice involves nontrivial operational challenges.

The first challenge is deciding on the extent to which the rate should be increased or decreased. The above theoretical rate-adjustment rule suffers from two serious problems: (a) The initial step size is potentially huge, resulting in a sender jumping between very low (e.g. 1 Mbps) and very high rates (e.g. 500 Mbps) in tens of milliseconds, causing high loss rates and latency inflation. (b) As time goes by, and θ diminishes, changes in rate become small, leading to slow reaction to changed network conditions like newly-free capacity. Second, there are challenges in the basic task of estimating γ. What happens when the environment is noisy, e.g., due to complex interactions between multiple senders, non-congestion loss, or microbursts unrelated to long-term congestion? We next explain how Vivace tackles this.

4.2 Translating Utility Gradients to Rates

Vivace's online learning algorithm begins by computing the gradient of the utility function. Suppose the current sending rate is r. Then, in the next two MIs, the sender will test the rates $r(1+\varepsilon)$ and $r(1-\varepsilon)$, compute the corresponding numerical utility values, u_1 and u_2, respectively, and estimate the gradient of the utility function to be $\gamma = \frac{u_1 - u_2}{2\varepsilon r}$. Then, 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, 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 $\Delta_r = \theta_0 \gamma$ (i.e., $r_{\text{new}} = r + \theta_0 \gamma$). 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 sender makes τ consecutive decisions to change the rate in the same direction, θ is set to $m(\tau)\theta_0$ (and so the rate is changed by $\Delta_r = m(\tau)\theta_0\gamma$). Setting $m(0) = 1$ implies that initially the change in rate is $\theta_0\gamma$, as described above. When the direction at which rate is adapted is reversed (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 large changes to network conditions between MIs. For instance, a burst of losses when probing $r(1-\varepsilon)$ and no losses when probing $r(1+\varepsilon)$ might result in huge γ and, consequently, a drastic rate change that overshoots the link's capacity. To address this, we introduce a mechanism, called

the *dynamic change boundary* ω. Whenever Vivace's computed rate change (Δ_r) exceeds ωr, the effective rate change is capped at ωr. 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 $\omega = \omega_0 + k \cdot \delta$ 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 $\Delta_r \leq r \cdot \omega$, Vivace recalibrates the value of k in the formula $\omega = \omega_0 + k \cdot \delta$ 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).

4.3 Contending with Unreliable Statistics

In general, accurate measurements require long observation time, yet that slows reaction to changing environments. We next discuss the ideas Vivace incorporates to address this challenge.

Estimating the RTT gradient via linear regression. The RTT gradient $\frac{d(RTT_x)}{dt}$ in MI x could be estimated by quantifying the RTT experienced by the first packet and the last packet sent in that MI. To estimate the RTT gradient more accurately, we utilize linear regression. Vivace assembles the 2-dimensional data set of (sampled packet RTT, time of sampling) for the packets in a MI, 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, e.g., because of recovery from packet losses in the physical layer (especially on wireless links), software packet processing devices, forwarding path flaps, or simply measurement errors due to processing time at the end-host networking stack. When Vivace employs a latency-sensitive utility function, this can result in misinformed decisions. To resolve this, Vivace leverages a low-pass filtering mechanism that treats latency gradient measurement smaller than $flt_{latency}$ as 0, to ignore small, brief latency jitters.

Double checking abnormal measurements. Occasionally, measurements lead to "counterintuitive" observations. We address the specific case that sending faster results in lower loss. While in general we avoid assumptions about the network, even with complex conditions it is highly unlikely that sending faster is the *cause* of lower loss; more likely, this is due to measurement noise or changing conditions (e.g., another sender reducing its rate). In this abnormal situation, Vivace "double checks" by re-running the same pair of rates. If this produces the same outcome in terms of which rate has higher utility, Vivace averages the utility-gradients; otherwise it throws out the original abnormal measurement.[2]

MI timeout. Generally, all information regarding packets sent 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 Vivace did before the sudden change are no longer meaningful. Therefore, if Vivace has not learned the fate of all packets sent in an MI when a certain timeout $T_{timeout}$ (a certain number of RTTs) expires, Vivace halves the sending rate.

4.4 TCP Friendliness

A common requirement from new congestion control schemes is to fairly share bandwidth with existing TCP connections (e.g., CUBIC). 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 [29], 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 the combination of Vivace's utility function and its rate control algorithm is a big step in this direction. Informally, Vivace 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. 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 com-

[2] In our experiments, double checking is mostly triggered during changing network conditions and multiflow competition, and lowers the packet loss rate with almost no influence on throughput, *e.g.*, turning on double-checking lowers Vivace's converged congestion loss from close to 7% to 5%. We omit these results due to limited space.

MI duration	1 RTT
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 $T_{timeout}$	4 RTT
confidence amplifier $m(\tau)$	τ $(\tau \leq 3)$ $2\tau - 3$ $(\tau > 3)$

Table 1: Vivace's rate control default parameters

pete against the TCP senders over the link capacity, effectively transforming into a loss-based protocol. Our experimental results in § 5.3 illustrate this intuition.

5 Implementation and Evaluation

We implemented a user-space prototype of Vivace based on UDT [14]. We set the specific parameters for t, b, c based on our theoretical analyses in § 3. We first set $t = 0.9$, satisfying the requirement that $t < 1$ in our family of utility functions, and then adjust the remaining parameters according to that t. We set $b = 900$ so as to achieve (in theory) no inflation in latency with up to 1000 competing senders on a 1000 Mbps bottleneck link, as established in Theorem 2. We set the parameter $c = 11.35$ so as to endure up to 5% random packet-loss rate according to the formula in §3.2 that $c = \frac{tC^{t-1}}{p}$. We believe these are reasonable design choices in practice, and note that our analysis enables tuning this parameter to accommodate other scenarios. Unless stated otherwise, our evaluation of Vivace uses the parameter default values in Table 1. For BBR, we use the *net-next* Linux kernel v4.10 [3].

Setup. We report on our experimental results with Vivace under emulated realistic network conditions, in the Internet, and in emulated application scenarios.

To cleanly separate Vivace's loss-related properties from its latency-related properties, our experiments sometimes involve evaluating Vivace when the latency penalty coefficient is $b = 0$, i.e., studying a purely loss-based variant of Vivace. We refer to this variant of Vivace as "*Vivace-Loss*" and to Vivace with the default parameter assignment as "*Vivace-Latency*".

5.1 Consistent High Performance

5.1.1 Resilience to Random Loss (Fig. 2)

Using Emulab [30], we evaluate the throughput of Vivace with a single flow on a link with 100 Mbps bandwidth, 30 ms RTT, 75 KB buffer, and varying random loss rate, and compare it with Allegro, BBR, and two TCP variants. As shown in Figure 2, both Vivace variants and Allegro achieve more than 90 Mbps throughput when the random loss rate is at most 3%, and remain above 80 Mbps until 3.5% loss rate. After that point, corresponding to the employed 5% loss resistance in the utility functions, their throughput reduces to $\frac{1}{10}$ of link

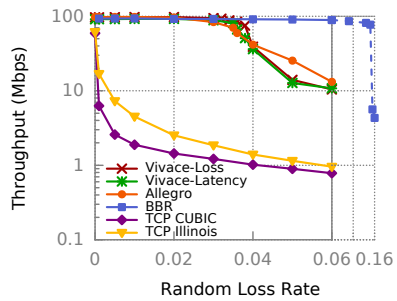


Figure 2: Random loss resilience

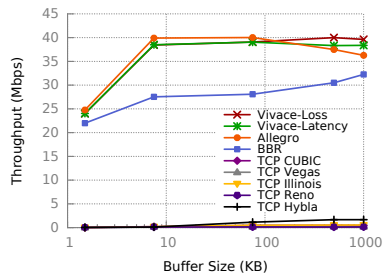


Figure 3: Long RTT tolerance

capacity. Vivace does not achieve full capacity at close to 4% random loss due to temporary bursty losses, which may exceed 5% in some monitor intervals.

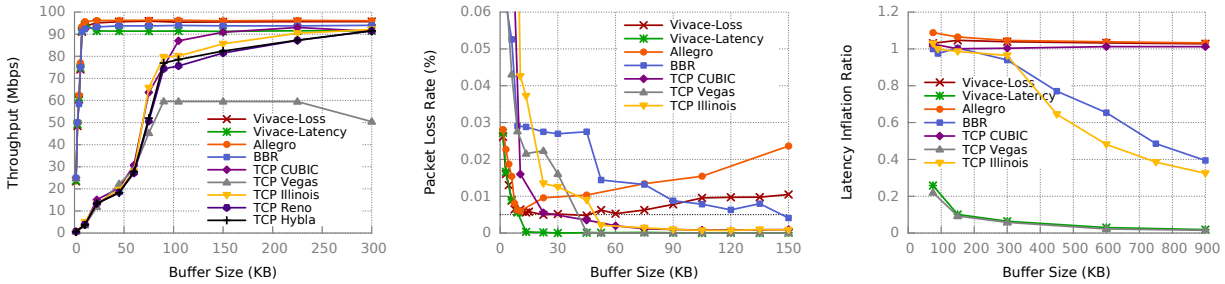
BBR keeps close-to-capacity throughput until 15% loss. By tuning Vivace's utility parameter c, we can achieve similarly high resilience to random loss. However, the theoretical insights (§ 3.2) and experiments we later present (§ 5.2.2) suggest that BBR's higher loss resilience induces comparable congestion loss with multiple competing flows, which we think is a less reasonable design choice. Finally, Figure 2 also shows gains of 20-50× over TCP family protocols.

5.1.2 High Performance on Satellite Links (Fig. 3)

We set up an emulated satellite link (as in [11]) with 42 Mbps bandwidth, 800 ms RTT and 0.74% random loss. Figure 3 shows the throughput achieved. Both Vivace-Loss and Vivace-Latency perform at least similar to Allegro, outperforming all the other protocols. Specifically, 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 Vivace flavors are at least 20% better than BBR as well, while the throughput of Allegro starts to fall. We also observed 20-300× higher performance compared to the best-in-class TCP variant.

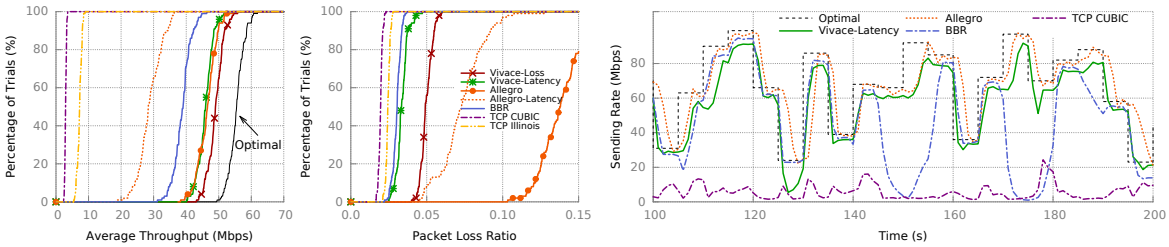
5.1.3 High Throughput without Bufferbloat (Fig. 4)

To demonstrate the effect of Vivace's latency-aware and provably fair utility function framework, we evaluate (using Emulab) its throughput and latency performance on a link with 100 Mbps bottleneck bandwidth, 30 ms RTT, and varying buffer size. On the throughput front, as



(a) High capacity (b) Low packet losses (c) Negligible RTT overflow

Figure 4: Vivace can achieve better performance with shallow buffer



(a) Achieving high capacity (b) Moderate packet loss (c) More responsive latency-sensitivity

Figure 5: Vivace can adapt to rapidly changing network conditions

shown in Figure 4(a), to achieve more than 90 Mbps, Vivace, BBR and Allegro only need a shallow queue of 7.5KB, which is 95% smaller than needed by CUBIC.

We next 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(b)[3], Vivace-Latency only needs a 13.5 KB buffer to guarantee nearly zero (less than 0.5%) loss, which is 55%, 70%, 74.3%, and 91% smaller than that of CUBIC, Vegas, Illinois, and BBR, respectively. Meanwhile, Vivace-Loss and Allegro exhibit interesting behavior as maximum buffer length grows. Initially, like all protocols, their loss rate decreases as the buffer becomes able to handle the inevitable randomness in packet arrival. But later, loss increases because a larger buffer (which these protocols fill) increases RTT and slows reaction when the buffer occasionally overflows. Even so, Vivace-Loss has lower loss than Allegro in all cases.

Finally, Figure 4(c) compares the latency inflation ratio, computed as the 95th percentile of self-inflicted RTT divided by the maximal possible latency inflation with given buffer size (when the buffer is full). Vivace-Latency'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 (i.e., an almost full buffer) for as large as 300 KB buffer. When the buffer size is as large as 2 BDP (750 KB), Vivace-Latency still has more than 90% less latency inflation ratio than BBR. BBR's performance disadvantage may be due to its white-box assumptions about the buffer size. Vegas achieves good performance on latency inflation by sacrificing the ability to fully utilize the available bandwidth (as shown in Figure 4(a)). As expected, Vivace-Loss, Allegro, and TCP CUBIC have around 100% inflation ratio, because they lack latency awareness.[4] In sum, Vivace achieves superior latency-awareness and high throughput at the same time.

5.1.4 Swift Reaction to Changes (Figures 5-6)

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

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

Figure 5(a) shows 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 Allegro, BBR, Allegro-Latency, TCP Illinois, and CUBIC, respectively. Vivace-Latency

[3]For conciseness, we leave out TCP Reno and Hybla, which have similar performance as the presented TCP variants.

[4]Although TCP CUBIC considers per-packet latency, it still cannot avoid severe buffer bloat, which explains its high inflation ratio.

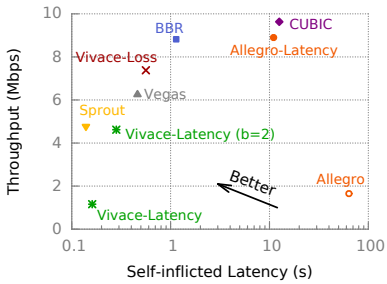


Figure 6: LTE throughput *vs.* self-inflicted latency

performs similarly to 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 Vivace's reactivity, we compare different protocols' packet loss. As shown in Figure 5(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 Allegro and Allegro-Latency by 55.7% and 75.8%. This is because Allegro's fixed-rate control algorithm reduces rate too slowly when available bandwidth suddenly decreases.

Figure 5(c) illustrates the behavior of several of the protocols across time. BBR occasionally suffers sudden rate degradation; we find this is associated with increases in latency, to which BBR reacts badly. Allegro reduces rate slower than Vivace-Loss when the bandwidth drops, which explains its higher packet loss rate.

LTE networks. An even more challenging network scenario, as suggested by [32, 36], 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 noise. On the other hand, if Vivace takes too long to measure, network conditions may have drastically changed, invalidating previous measurements.

We use Mahimahi [26] to replay the Verizon-LTE trace provided by [32]. We compare Vivace with Allegro-Latency, BBR, CUBIC, Vegas and Sprout [32]. Figure 6 shows the achieved tradeoff between throughput and self-inflicted latency (as defined in [32]). Allegro, due to its overly aggressive behavior, significantly inflates latency, and also fails to deliver good throughput. Vivace-Loss reduces latency by 50.7%, 94.9%, and 95.5% compared to BBR, Allegro-Latency, and TCP CUBIC, at the cost of only 16.3%, 17.1%, and 23.4% smaller throughput, respectively. However, Vivace is still suboptimal. Compared with the best-in-class TCP (Vegas), Vivace-Loss has 17.7% larger throughput, but also 21.6% larger latency. Sprout, which is specifically designed for cellular networks with an explicit cellular link measurement model and receiver-side feedback changes

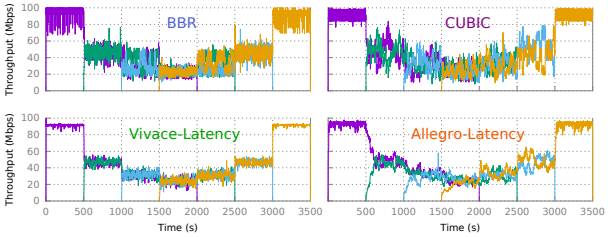


Figure 7: Vivace has fair and stable convergence

to TCP, outperforms Vivace-Loss with 75.2% shorter latency and only 35.6% smaller throughput. Furthermore, Vivace-Latency is impeded by the noisy latency measurements, and achieves the smallest throughput.

We also test with a smaller latency coefficient, $b = 2$. The consequence is supporting fewer competing senders, but the typical resource allocation in LTE networks [34] lowers the possibility of competition by many concurrent flows. This improves performance, with 39.0% lower latency and 26.3% smaller throughput than Vegas, but still falls short of Sprout. Vivace's performance in LTE networks may further improve through better reaction to noisy environments; we leave this to future work.

5.2 Convergence Properties

We next demonstrate that Vivace improves the convergence speed vs. stability tradeoff compared to state-of-the-art protocols. We also experimentally show the tradeoff between congestion loss and random loss resilience.

5.2.1 Convergence Speed and Stability (Fig. 7-8)

Temporal behavior of convergence. We set up a dumbbell topology on Emulab to demonstrate convergence performance with 4 flows sharing a link with 100 Mbps bandwidth, 30 ms RTT, and 75 KB buffer. Figure 7 shows the convergence process of several protocols with 1s granularity. 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, 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, reproducing an experiment in [11].

On a link of 100 Mbps bandwidth and 30 ms RTT, we let an initial flow run for 10 s, which is significantly longer than needed for its convergence, then start a second flow. The convergence time is calculated as the time from the second flow's entry to the earliest time after which it maintains a sending rate within ±25% of its ideal fair share (50 Mbps) for at least 5s. The convergence stability is calculated as the standard deviation of throughput of the second flow after its convergence.

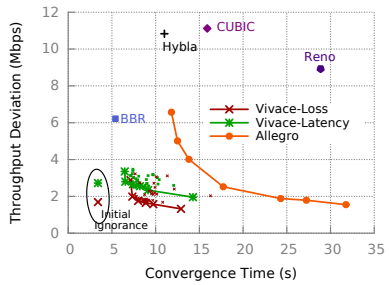


Figure 8: Better tradeoff curve

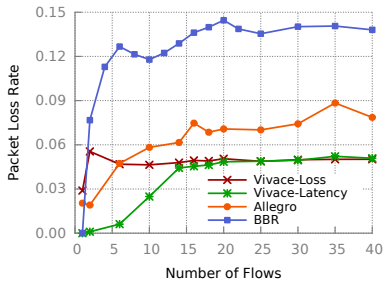


Figure 9: Multi-flow congestion

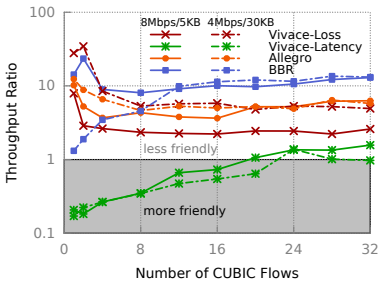


Figure 10: More friendly to TCP

To produce a tradeoff curve in Vivace, we vary parameters that affect response speed within a certain range ($1.0 \leq \theta_0 \leq 1.5$, $0.05 \leq \omega_0, \delta \leq 0.1$); we show all resulting points and highlight the lower-left Pareto front.

Figure 8 illustrates that there is a "virtual wall" in the trade-off plane at around 10s convergence time that neither TCP variants nor Allegro can pass even by trading off stability. Interestingly, BBR and Vivace penetrate that wall. Vivace, by default, achieves significantly better tradeoff points. Both Vivace-latency and Vivace-loss achieve similar convergence stability as Allegro, but using nearly 60% smaller convergence time. With convergence speed slightly higher than BBR, both Vivace variants have around 50% smaller throughput deviation. These improved trade-offs demonstrate the effectiveness of Vivace's no-regret online learning scheme.

However, any of the protocols here could choose to adopt more aggressive slow-start algorithms. As a simple example, we implemented an "initial ignorance" of loss (first 50 lost packets) and latency inflation (gradient smaller than 0.2), resulting in 37% faster convergence, and better stability, than BBR. In general, the startup algorithm is orthogonal to long-term rate control, and more advanced techniques like [22] might improve both BBR and Allegro; we leave this to future work.

5.2.2 Lower Price for Loss Resilience (Figure 9)

As we analyzed in § 3.2, 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 within the Vivace framework. Due to the network dynamics from multi-sender interaction and efficiency of rate control algorithm, the actual price can be even higher.

To experimentally evaluate this trade-off, we set up an experiment with 30 ms RTT. We increase the number of concurrent competing flows, while proportionally increasing the FIFO queue link's total bandwidth to maintain a *per flow* 8 Mbps and 25 KB (close to 1 BDP) buffer share on average. Figure 9 shows the average packet loss per flow as the number of flows 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 Vivace's online learning analysis framework, it shows a surprisingly similar tradeoff: it endures more random loss but pays a much higher price (14% loss) compared to 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 a lag in interactive or video streaming applications; and the large amount of transmission can cause additional energy burden on mobile devices.

In light of this discovery, we urge future congestion control designs to carefully consider this tradeoff. Though Allegro also achieves 5% random loss resilience similar to Vivace, due to its naïve rate control algorithm, it pays a higher convergence loss (9%) price. BBR, positioned as a latency-aware protocol, also has the same effect. 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 number of concurrent senders increases. This may be due to factors not modeled in our theoretical analysis, including measurement noise and the fact that the large number of senders are constantly probing rather than staying in a perfect equilibrium. We hope to study this congestion loss in the future.

5.3 Improved Friendliness to TCP

We set up a 30 ms RTT bottleneck link with one flow using a new protocol (BBR, Allegro, or Vivace) and competing with an 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, as before. We used two per-flow share settings: (4Mbps, 30KB) and (8Mbps, 5KB), corresponding to 2BDP and 0.12BDP of buffer size, respectively. Figure 10 shows the ratio between the throughput of the new-protocol flow and average throughput per CUBIC flow. A ratio of 1 in Figure 10 indicates perfect friendliness; larger ratios indicate aggressiveness towards CUBIC.

Vivace-Latency behaves as expected in the design (§4.4). When the number of CUBIC flows is small, since the queue is not always full, Vivace-Latency finds it can

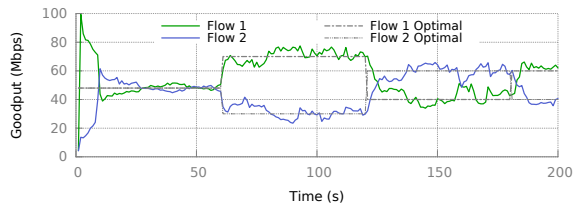


Figure 11: Flexible equilibrium by tuning utility knobs

reduce RTT by reducing its rate, and thus achieves lower throughput than CUBIC flows. However, as the number of CUBIC senders increases, it achieves the best fairness among new generation protocols. 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 §5.5.2 strongly suggest positive results.

Among the evaluated protocols, BBR yields the worst TCP friendliness. In [1, 8], BBR's TCP friendliness was noted to be satisfactory, based on a single BBR flow competing with a single CUBIC flow over a large buffer (2 BDP). We successfully reproduced this specific result (the leftmost point in the 4 Mbps/30 KB (2 BDP) BBR line of Figure 10). 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 with the number of CUBIC flows until about 16. Therefore, in practice, BBR can be very unfriendly when there are multiple competing CUBIC flows.

Though Allegro and Vivace-Loss are both loss-based, Vivace is much more agile in its reaction to competing TCP flows, especially under a shallow buffered network: its throughput ratio converges at 2.5 *vs.* Allegro's 8. In addition, though Vivace-Loss dominates CUBIC when the number of concurrent flows is small, the final converged throughput ratio (about 5 at 2 BDP) is less than half of that in BBR. In sum, though perfect TCP friendliness is fundamentally hard, we believe that Vivace provides a viable path towards adoption.

5.4 Flexible Convergence Equilibrium

With its unique utility function framework (§ 3.3), Vivace unleashes the potential to be flexible and centrally controlled. To demonstrate this capability experimentally, we set up a link with 100 Mbps bandwidth, 30 ms RTT, and two competing flows. As shown in Figure 11, 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 illustrate the basic capability of Vivace; we leave a full-fledged system leveraging this capability to future work.

5.5 Benefits in the Real World

In addition to the above transport-level experiments on controlled networks, we test a video streaming application and performance in the wild Internet.

5.5.1 Video Streaming

We implemented a transparent proxy so that RTSP-over-TCP traffic flows from an OpenRTSP [4] client over a legacy TCP connection ending at a client-side proxy, then over a configurable transport protocol across the bottleneck link to a server-side proxy, and finally over a second legacy TCP connection to an OpenRTSP server; similar proxying occurs in reverse. We compare the streamed video's buffering ratio [10], calculated as the ratio of time spent during buffering relative to total streaming session time, using Vivace-Latency, Allegro-Latency, BBR, and TCP CUBIC. We test with four 4K videos, with 15 Mbps, 30 Mbps, 50 Mbps, and 95 Mbps average bit rate requirements in experiments in Emulab.

We first evaluate the buffering ratio with RTT changing every five seconds to a value uniform-randomly selected between 10 ms and 100 ms. We set up a link with 300 KB buffer and 0.01% random loss rate, with the network bandwidth at least 10% more than the bit rate required to stream the requested video. Fig. 12(a) shows the average buffering ratio of Vivace-Latency stays below 8%, similar to Allegro-Latency – a reduction of at least 86% and 90% compared with BBR and CUBIC.

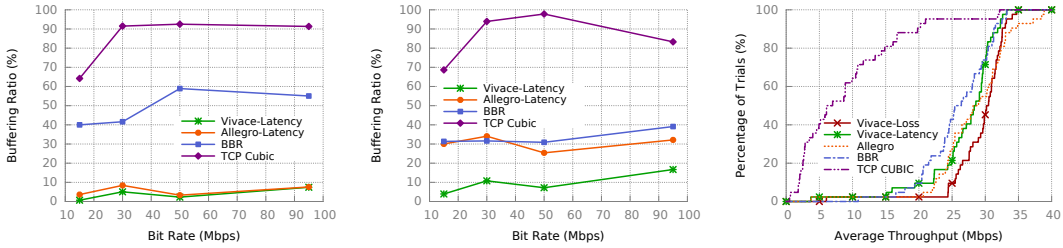
To demonstrate the application-level benefit of Vivace's stable convergence, we set up three competing streaming flows from three client-server pairs. They share a bottleneck link with 75 KB buffer, 100 ms RTT, 0.01% random loss, and adequate bandwidth for all three to stream. As shown in Figure 12(b), Vivace-Latency outperforms Allegro-Latency, BBR, and CUBIC by at least 48%, 57%, and 80%, respectively. We attribute the degraded performance of Allegro-Latency to its inferior latency awareness and reaction, and that of BBR to its high throughput variance among flows.

5.5.2 The Wild Internet

Finally, we evaluate 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.*, 42 sender-receiver pairs.[5] 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.[6] For each AWS site we test all protocols, and compute the average throughput of each protocol from five 100 sec transmissions. Figure 12(c)

[5] We test only the uplink because the virtualized AWS server impacts performance of our current user-space UDP-based packet pacing.

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



(a) Video streaming with varying latency

(b) Multiflow video streaming

(c) Throughput from home networks to AWS

Figure 12: Performance gain in application and live Internet environments

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 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 the median, *i.e.*, CUBIC flows just cannot efficiently utilize the available bandwidth. This serves as a strong validation that Vivace provides a viable deployment path, although larger scale evaluation is desirable. Specifically, more substantial measurements are needed to answer the question of how Vivace behaves on real-world traffic patterns [28].

The root cause of the difference between Vivace-Loss and Allegro is 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 loss rate and often yield more aggressive behavior compared to Vivace.

6 Related Work

We have already discussed Remy, and compared with several TCP variants, PCC Allegro, and BBR. We next place Vivace in the context of other related work.

In-network feedback. One class of protocols improve congestion control by providing explicit in-network feedback (e.g. available bandwidth and ECN) for better informed decisions [5, 7, 20]. These protocols yield good performance, but have been proven to be hard to deploy outside of data centers: they require coordinated change of protocols and network devices. Vivace on the other hand is compatible with the TCP message format, 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 [32, 36] or data center networks [5, 24, 33] 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 [24] uses the RTT-gradient as a control signal similar to Vivace, but it still uses a hardwired rate control algorithm that maps events to fixed reactions (e.g., using fixed thresholds and step sizes). Protocols in this class provide significant performance gains, but only target very specific environments. Some of them also require changes at both endpoints [32], which may challenge deployment in practice.

Short flows. Some congestion control protocols aim to optimize performance of very short flows [22, 25]. These are complementary to Vivace, 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 feedback) whereas Vivace targets the "closed loop" phase of data transfer (when meaningful feedback can be gathered with long enough data transfer). In fact Vivace could plausibly utilize [22, 25] as its starting phase, replacing slow-start.

7 Conclusion

We proposed Vivace, a congestion control architecture based on online optimization theory. 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 Vivace significantly improves upon the existing state of the art in terms of performance, convergence speed, reactiveness, TCP friendliness, and more. Vivace requires sender-only changes and is hence readily deployable. We leave research questions regarding centralized resource allocation via Vivace's simple interface, and Vivace's integration into comprehensive emulators such as Pantheon [35] and production systems such as QUIC [21] and the Linux kernel, to the future.

We thank our shepherd, Alex Snoeren, and the reviewers for their helpful comments, and Google and Huawei for ongoing support of the PCC project.

References

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

[2] Full Proof of Theorems. http://www.ttmeng.net/pubs/vivace_proof.pdf.

[3] Linux net-next. https://kernel.googlesource.com/pub/scm/linux/kernel/git/davem/net-next.git/+/v4.10.

[4] OpenRTSP. http://www.live555.com/openRTSP/.

[5] ALIZADEH, M., GREENBERG, A., MALTZ, D., PADHYE, J., PATEL, P., PRABHAKAR, B., SENGUPTA, S., AND SRIDHARAN, M. Data center TCP. *Proc. of ACM SIGCOMM* (September 2010).

[6] BRAKMO, L., LAWRENCE, S., O'MALLEY, S., AND PETERSON, L. TCP Vegas: New techniques for congestion detection and avoidance. *Proc. of ACM SIGCOMM* (1994).

[7] CAESAR, M., CALDWELL, D., FEAMSTER, N., REXFORD, J., SHAIKH, A., AND VAN DER MERWE, K. Design and implementation of a routing control platform. *Proc. of NSDI* (April 2005).

[8] CARDWELL, N., CHENG, Y., GUNN, C., YEGANEH, S., AND JACOBSON, V. BBR: Congestion-based congestion control. *Queue 14*, 5 (2016), 50.

[9] CHOWDHURY, M., ZHONG, Y., AND STOICA, I. Efficient coflow scheduling with varys.

[10] DOBRIAN, F., SEKAR, V., AWAN, A., STOICA, I., JOSEPH, D., GANJAM, A., ZHAN, J., AND ZHANG, H. Understanding the impact of video quality on user engagement. *Proc. of ACM SIGCOMM* (August 2011).

[11] DONG, M., LI, Q., ZARCHY, D., GODFREY, P. B., AND SCHAPIRA, M. PCC: Re-architecting Congestion Control for Consistent High Performance. *Proc. of NSDI* (March 2015).

[12] EVEN-DAR, M. E., MANSOUR, Y., AND NADAV, U. On the convergence of regret minimization dynamics in concave games. *Proc. of ACM symposium on Theory of computing* (2009).

[13] GHOBADI, M., YEGANEH, S., AND GANJALI, Y. Rethinking end-to-end congestion control in software-defined networks. *Proc. of HotNets* (November 2012).

[14] GU, Y. *UDT: a high performance data transport protocol*. University of Illinois at Chicago, 2005.

[15] HA, S., RHEE, I., AND XU, L. CUBIC: A new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review* (2008).

[16] HAZAN, E. Introduction to online convex optimization. http://ocobook.cs.princeton.edu/OCObook.pdf.

[17] HONG, C., KANDULA, S., MAHAJAN, R., ZHANG, M., GILL, V., NANDURI, M., AND WATTENHOFER, R. Achieving high utilization with software-driven WAN. *Proc. of ACM SIGCOMM* (August 2013).

[18] JAIN, S., KUMAR, A., MANDAL, S., ONG, J., POUTIEVSKI, L., SINGH, A., VENKATA, S., WANDERER, J., ZHOU, J., AND ZHU, M. B4: Experience with a globally-deployed software defined wan. *ACM Computer Communication Review* (September 2013).

[19] JIANG, J., SUN, S., SEKAR, V., AND ZHANG, H. Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation. *Proc. of NSDI* (March 2017).

[20] KATABI, D., HANDLEY, M., AND ROHRS, C. Congestion control for high bandwidth-delay product networks. *Proc. of ACM SIGCOMM* (August 2002).

[21] LANGLEY, A., RIDDOCH, A., WILK, A., VICENTE, A., KRASIC, C., ZHANG, D., YANG, F., KOURANOV, F., SWETT, I., IYENGAR, J., ET AL. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (2017), ACM, pp. 183–196.

[22] LI, Q., DONG, M., AND GODFREY, P. Halfback: Running short flows quickly and safely. *Proc. of CoNEXT* (November 2015).

[23] LIU, S., BAŞAR, T., AND SRIKANT, R. TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks. *Performance Evaluation* (2008).

[24] MITTAL, R., LAM, V. T., DUKKIPATI, N., BLEM, E. R., WASSEL, H. M. G., GHOBADI, M., VAHDAT, A., WANG, Y., WETHERALL, D., AND ZATS, D. TIMELY: RTT-based congestion control for the datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015* (2015), S. Uhlig, O. Maennel, B. Karp, and J. Padhye, Eds., ACM, pp. 537–550.

[25] MITTAL, R., SHERRY, J., RATNASAMY, S., AND SHENKER, S. Recursively cautious congestion control. *Proc. of NSDI* (March 2014).

[26] NETRAVALI, R., SIVARAMAN, A., DAS, S., GOYAL, A., WINSTEIN, K., MICKENS, J., AND BALAKRISHNAN, H. Mahimahi: Accurate record-and-replay for HTTP. *Proc. USENIX ATC* (August 2015).

[27] SIVARAMAN, A., WINSTEIN, K., THAKER, P., AND BALAKRISHNAN, H. An experimental study of the learnability of congestion control. *Proc. of ACM SIGCOMM* (August 2014).

[28] SUN, Y., YIN, X., JIANG, J., SEKAR, V., LIN, F., WANG, N., LIU, T., AND SINOPOLI, B. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. *Proc. of ACM SIGCOMM* (August 2016).

[29] WEI, D., JIN, C., LOW, S., AND HEGDE, S. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Transactions on Networking* (2006).

[30] WHITE, B., LEPREAU, J., STOLLER, L., RICCI, R., GURUPRASAD, G., NEWBOLD, M., HIBLER, M., BARB, C., AND JOGLEKAR, A. An integrated experimental environment for distributed systems and networks. *Proc. of OSDI* (December 2002).

[31] WINSTEIN, K., AND BALAKRISHNAN, H. TCP ex Machina: computer-generated congestion control. *Proc. of ACM SIGCOMM* (August 2013).

[32] WINSTEIN, K., SIVARAMAN, A., AND BALAKRISHNAN, H. Stochastic forecasts achieve high throughput and low delay over cellular networks. *Proc. of NSDI* (March 2013).

[33] WU, H., FENG, Z., GUO, C., AND ZHANG, Y. ICTCP: Incast congestion control for TCP in data center networks. *Proc. of CoNEXT* (November 2010).

[34] XIE, X., ZHANG, X., KUMAR, S., AND LI, L. E. pistream: Physical layer informed adaptive video streaming over lte. In *Mobicom* (2015).

[35] YAN, F. Y., MA, J., HILL, G., RAGHAVAN, D., WAHBY, R. S., LEVIS, P., AND WINSTEIN, K. Pantheon: the training ground for internet congestion-control research, 2018.

[36] ZAKI, Y., PÖTSCH, T., CHEN, J., SUBRAMANIAN, L., AND GÖRG, C. Adaptive congestion control for unpredictable cellular networks. *Proc. of ACM SIGCOMM* (August 2015).

[37] ZINKEVICH, M. Online convex programming and generalized infinitesimal gradient ascent. In *ICML* (2003), AAAI Press, pp. 928–936.