ETC: An Elastic Transmission Control Using End-to-End Available Bandwidth Perception

Feixue Han, *Tsinghua Shenzhen International Graduate School and Peng Cheng Laboratory;* Qing Li, *Peng Cheng Laboratory;* Peng Zhang, *Tencent;* Gareth Tyson, *Hong Kong University;* Yong Jiang, *Tsinghua Shenzhen International Graduate School and Peng Cheng Laboratory;* Mingwei Xu, *Tsinghua University;* Yulong Lan and ZhiCheng Li, *Tencent*

https://www.usenix.org/conference/atc24/presentation/han

This paper is included in the Proceedings of the 2024 USENIX Annual Technical Conference.

July 10–12, 2024 • Santa Clara, CA, USA

978-1-939133-41-0

Open access to the Proceedings of the 2024 USENIX Annual Technical Conference is sponsored by

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

ETC: An Elastic Transmission Control Using End-to-End Available Bandwidth Perception

Feixue Han
Tsinghua Shenzhen International Graduate School
Peng Cheng Laboratory

Qing Li
Peng Cheng Laboratory

Peng Zhang
Tencent

Gareth Tyson
Hong Kong University

Yong Jiang
Tsinghua Shenzhen International Graduate School
Peng Cheng Laboratory

Mingwei Xu
Tsinghua University

Yulong Lan
Tencent

ZhiCheng Li
Tencent

Abstract

Researchers and practitioners have proposed various transport protocols to keep up with advances in networks and the applications that use them. Current Wide Area Network protocols strive to identify a congestion signal to make distributed but fair judgments. However, existing congestion signals such as RTT and packet loss can only be observed *after* congestion occurs. We therefore propose Elastic Transmission Control (ETC). ETC exploits the instantaneous receipt rate of N consecutive packets as the congestion signal. We refer to this as the *pulling rate*, as we posit that the receipt rate can be used to "pull" the sending rate towards a fair share of the capacity. Naturally, this signal can be measured prior to congestion, as senders can access it immediately after the acknowledgment of the first N packets. Exploiting the pulling rate measurements, ETC calculates the optimal rate update steps following a simple elastic principle: the further away from the pulling rate, the faster the sending rate increases. We conduct extensive experiments using both simulated and real networks. Our results show that ETC outperforms the state-of-the-art protocols in terms of both throughput (15% higher than Copa) and latency (20% lower than BBR). Besides, ETC shows superiority in convergence speed and fairness, with a 10× improvement in convergence time even compared to the protocol with the best convergence performance.

1 Introduction

Recent years have seen a resurgence of work in **C**ongestion **C**ontrol (CC) [1–5]. This has been driven by the increasing diversification of applications (*e.g.*, web and video streaming [6–8]) and networked environments (*e.g.*, 5G). These algorithms rely on congestion signals — primarily loss, delay, and **E**xplicit **C**ongestion **N**otifications (ECN) — to manage their sending rate. All tend to have the common goal of attaining high throughput, low latency, and fairness. However, in practice, this is difficult because these requirements often conflict. This mandates a strategic trade-off, driven by two key observations.

The *first observation* is that congestion control algorithms, which prioritize high throughput, often struggle to simultaneously attain low latency [1, 3, 4, 9, 10]. This is because such algorithms usually rely on *loss signals*, thereby becoming less sensitive to additional queuing delays. Even BBR, which tries to limit the inflight bytes within a **B**andwidth **D**elay **P**roduct (BDP), still fails to avoid high latency, due to its excessive estimation of **A**vailable **B**and**w**idth (ABW) [11]. This particularly occurs when there are multiple flows or a long waiting period for packet evacuation (minimum **R**ound **T**rip **T**ime (RTT) probing). In contrast, algorithms that prioritize delay [2, 5, 12, 13] show favorable latency performance but attain lower throughput. This is because they fail to compete with other more aggressive flows. For example, Vegas [12] and Fast [13] fail to ensure bandwidth share when competing with buffer-filling flows. Further, Copa [2] and Vivace [5] obtain poor bandwidth utilization even when there are no other competing flows.

The *second observation* is that current algorithms only consider fairness *after* flows have fully occupied the bandwidth, or even after packet loss has occurred [1–5, 9, 10, 14]. This means they often occupy an unfair position, leading to bad inter-flow fairness. Specifically, these algorithms manage their sending rates via slow start in the start-up stage. However, slow start's exponential increases can result in a large disparity among asynchronous flows, since the flows that start first have a better chance to preempt the bandwidth. On the basis of this large disparity, convergence can take tens of seconds (or even fail). For example, CUBIC [9] has to wait for a packet loss event to free up bandwidth, and PCC [4] nearly always fails to achieve fairness with its **M**ultiplicative **I**ncrease **M**ultiplicative **D**ecrease (MIMD) mechanism [15]. Therefore, shorter flows may complete their transmissions without obtaining a fair bandwidth share, since it takes such a long time to achieve convergence.

We argue that the above two observations are driven by the inherent limitations of *only* using reactive congestion signals

Corresponding Author: Qing Li, liq@pcl.ac.cn

as the network feedback. This is because traditional congestion signals only work when the collective flows' rates exceed the link capacity. When the network bottleneck capacity is not occupied, it is difficult for senders to reach a rapid consensus on their fair share of the ABW. With this in mind, we introduce the concept of a *pre-congestion consensus signal*: A shared signal that can be witnessed by all senders to reach an agreement on the ABW for their respective flows before congestion occurs. This signal should be (*i*) easy to obtain, without adding significant overhead; (*ii*) common across all senders, such that all parties sharing a bottleneck link obtain the similar signal value; and (*iii*) proactive, with the ability to be created prior to congestion being observed. We argue that such a signal could then be used by all senders to rapidly converge on the appropriate sending rate without having to wait for a congestion event to occur.

In this paper, we leverage a lightweight ABW measurement mechanism to obtain such a signal, which we refer to as the *pulling rate*. Put simply, the pulling rate is the instantaneous receipt rate of $N (N \geq 2)$ consecutive packets (denoted as a *micro-burst*). Note that the idea of measuring the ABW by collecting the instantaneous receipt rate of N consecutive packets has been widely used in previous studies [16–21]. By sharing this common measurement methodology and continuous refinement of the results, all senders sharing the same bottleneck will obtain similar pulling rate signals. In contrast to congestion control algorithms that rely on reactive congestion signals (*e.g.*, packet loss), the available bandwidth measured via the pulling rate can then serve as a constant guide for all competing flows to set their sending rates, even before congestion has occurred.

We embed these concepts into a new congestion control algorithm: **E**lastic **T**ransmission **C**ontrol (ETC), which exploits a combination of the pulling rate and RTT measurements to identify and adapt to congestion conditions. Upon a new flow being initiated, ETC begins to continually measure the pulling rate. In parallel, it also measures the RTT to infer if there is potentially excess capacity on the path. In cases when the RTT suggests there is spare bandwidth, the ETC sender increases (pulls) its sending rate towards the pulling rate in an attempt to utilize excess capacity.

The above design raises three practical challenges though. The *first challenge* is how to rapidly calculate the optimal pulling rate at the beginning of a transmission, without perturbing the network. Conventional ABW probing methods [22, 23] necessitate the utilization of supplementary probing packet pairs or packet trains. To mitigate disruptions to the network, we directly pace data transmission within micro-bursts and collect the instantaneous receipt rate of each micro-burst. We then continuously correct the pulling rate with the maximum average receiving rate of the subsequent traffic, such that all flows can reach a consensus on the pulling rate.

The *second challenge* is the insufficient accuracy of the current system timers. This is not a problem in traditional CC logic, which relies on coarser metrics like packet loss. However, it is problematic for ETC because we pace data transmission on granularity as little as 2 packets (*i.e.* per micro-burst) to calculate the pulling rate. The benefit is that it gives ETC fine-grained control over the sending rate. However, the necessary timing cannot be implemented on current commodity systems, as their timers are too coarse-grained. To address this, ETC proposes the use of **ack**nowledgment (ACK)-clocking, which exploits the arrival of ACKs to pace data transmission, thereby compensating for insufficient timer precision.

The *third challenge* is how we can use the pulling rate to dynamically select the optimal step size. Even with the prior knowledge of available bandwidth, the indeterminate number of concurrent flows makes it challenging to determine how the sending rate should be adjusted upon each iteration. Most prior works rely on a fixed step size technique. However, this design (*i*) fails to adapt to different networks, since the step size is not related to the available bandwidth; and (*ii*) cannot guarantee the flow rates move towards fair equilibrium (*e.g.*, aggressive exponential step sizes can result in a growth in inter-flow unfairness). To address this, ETC introduces the use of a concave function that maps the distance between the pulling rate and the current sending rate to guide rate adjustments. This concave formula avoids exceeding the fair bandwidth share and we prove that it reduces the difference among the flows' rates at each adjustment, thereby accelerating the convergence toward fairness.

We have a fully deployed user-space implementation of ETC, which has served as the transmission protocol for our commercial video application for over a year. During this period, ETC has served millions of daily users. Using this implementation, we conduct comprehensive experiments in both emulated and real-world international/ intercontinental Internet scenarios to demonstrate the performance of ETC versus other state-of-the-art protocols. Our key results are as follows:

- ETC achieves the highest throughput under almost all scenarios considered. We simultaneously attain the lowest latency. ETC obtains up to 15% higher throughput than Copa on short paths and 18% more than CUBIC on long paths.
- ETC attains the best latency. It achieves a 10% and 18% lower 95th latency than BBR under one-flow and three-flow scenarios. For the short paths, ETC obtains more than 10% lower latency compared to almost all competing algorithms.
- ETC attains superiority in convergence speed and fairness. ETC can achieve fair convergence within $2s$ after a new flow starts. CUBIC, with the second-best convergence, takes more than 20s to converge.

Although finer-grained timers are available in data center hardware, we target commodity devices, *e.g.*, laptops, smart phones.

2 Background and Motivation

In this section, we highlight the factors that hinder current transmission protocols from achieving better performance. We discuss the deficiencies of state-of-the-art protocols and propose key directions for improvement.

2.1 Limitations of Congestion Signals

In **W**ide **A**rea **N**etworks (WAN), loss and RTT are the two mainstream signals used by most congestion control algorithms. For example, as a loss-oriented protocol, PCC [4] evaluates its current sending rate against its measured loss rate. Similarly, CUBIC [9] reduces its rate in the case of packet loss, considering the current window size the maximum. In contrast, as RTT-oriented protocols, Vegas [12] and Copa [2] attempt to limit the RTT within a preset range through rate adjustment. In an attempt to balance these two signals, Vivace [5] constructs a utility function with both the delay and loss rate to evaluate the current sending rate.

Unfortunately, exclusively using these reactive signals has a key limitation: they work only once the collective flows' rates exceed the link capacity. Specifically, RTT will be minimal until packets begin to accumulate in the router buffers. Eventually, a loss event will indicate severe congestion. Prior to this point, loss-oriented protocols will assume there is still spare bandwidth, while RTT-oriented protocols will assume there is congestion. That is, individual flows may perform too conservatively or aggressively, resulting in either bandwidth waste or congestion. Moreover, this postpones the time when flows converge. We argue that an effective mechanism should employ both reactive signals (such as RTT) and more proactive signals that can be sensed prior to link capacity being saturated. To achieve fast convergence and fairness, we seek a signal that not only obtains consensus among flows, but also is capable of guiding rate increases.

2.2 Limitations of Current Algorithms

Alongside the limitations of the congestion signals, current congestion control algorithms suffer from several issues.

Lack of Fair Convergence. The correct number of inflight bytes is the product of a fair bandwidth share and the base RTT. However, current algorithms show poor performance in fair convergence due for two reasons: (*i*) They only consider convergence after the flows have fully occupied the available bandwidth; and (*ii*) Their rate increase/decrease step size is not related to the available bandwidth. We next illustrate these two points in detail.

In the start-up stage, all current algorithms begin with a slow start. Slow start increases the flow rate exponentially. It thus enlarges the rate difference among flows that are started asynchronously. Following this, flows attempt to reach fair convergence, *i.e*. the rate adjustments gradually reduce the rate difference among competing flows. During this period, the convergence speed depends on the choice of step size. The step sizes in current algorithms can be divided into two categories: (*i*) dynamic step sizes based on the flow's own rate [1, 5, 9] (faster flows with smaller step size); or (*ii*) fixed step sizes [2–4, 10, 12]. Although dynamic step sizes can promote fairness among flows (compared with the fixed step size) since slower flows have more chance to compete for bandwidth, all of these step sizes are independent of the available bandwidth. This severely delays the convergence, without fine-tuning specific network environments. We analyze how the current algorithms guarantee their convergence in Appendix C. To summarize, to obtain fast and fair convergence, we need a dynamic step size that adapts to the available bandwidth.

Lack of Optimal Rate Change. It is necessary for CC algorithms to change the sending rate appropriately. Intuitively, the rate change should become smaller as it approaches the available bandwidth. However, due to the lack of an effective estimation of the available bandwidth, current algorithms adopt an overly conservative step size at first while becoming more aggressive in the subsequent adjustments [2, 4, 5]. This is particularly severe in the slow start phase, where the flows expand the rate in an exponential manner [24]. This drastic rate change can excessively overshoot the available bandwidth. Several more recent protocols [2, 4, 5] adopt a velocity parameter to speed up the convergence. That is, the flows update their rate faster in the face of continuous rate increases/decreases. We argue this violates the intuition that the closer to the target rate, the smaller the update should be. These frequent and intense rate oscillations magnify the probability of bandwidth overuse or waste.

2.3 Our Design Principles

Based on the above observations, we advocate two novel design principles to improve congestion control performance in terms of throughput, delay, and fair convergence simultaneously.

Pre-Congestion Consensus. First, we argue that it is necessary to identify and propose a pre-congestion consensus signal. Recall, this is a signal that can be used by all senders (sharing a bottleneck) to reach a common estimate of the available bottleneck bandwidth. Importantly, this consensus must be reachable *before* packets begin to accumulate at the in-network bottleneck (so as to guide rate increases). Note, present congestion signals (*e.g*., delay and packet loss) fail to meet these requirements. Equally, using ECN is not practical outside of a data center network, and leaves open the challenge of selecting when a router should trigger an ECN.

Dynamic Step Sizes. Second, we argue that the congestion control algorithms should employ dynamic step sizes to meet the following requirements: (*i*) *Safe*: the rate increases become less aggressive as they approach the available band-

width so that the sending rate does not excessively exceed the available bandwidth. (*ii*) *Fast*: quickly seizing the spare bandwidth while quickly converging to an equilibrium. (*iii*) *Fair*: giving slower flows more chance to compete for the available bandwidth and punishes faster flows more strictly.

3 Rate Control in ETC

We start by explaining how ETC senders set their sending rate. Algorithm 1 presents an overview of the algorithm.

3.1 Overview of Rate Control

Figure 1 shows an overview of ETC. We provide a summary below of how a sender decides when to accelerate or decelerate its sending rate.

Rate Acceleration. When the RTT *holds or decreases* (the left blue part), the sender infers that there may be spare bandwidth available. Despite this, it does not know how much bandwidth is available (since there is no explicit signal to show the link utilization). Therefore, to guide the sender, we propose that the sender calculates the *pulling rate* (R_p). This is an estimate of the bandwidth, which is measured using a lightweight passive ABW measurement mechanism at the sender. Put simply, it is the instantaneous receipt rate of $N(N \geq 2)$ consecutive packets (denoted as *micro-burst*), constantly corrected by the maximum average receiving rate. Upon detecting the possibility of spare capacity, the sender uses the pulling rate to calculate the precise rate change. A key innovation is the introduction of elasticity: the further away from the available bandwidth, the faster the sending rate (s) increases, and the slower the sending rate decreases. We provide further details in Section 3.3.

Rate Deceleration. When the RTT gradient exceeds a preset threshold (the right purple part), ETC assumes that it is approaching (or at) the maximum bandwidth allocation. At this stage, the sender temporarily regards the current receiving rate (r) as the optimal rate. Thus, the sender gradually reduces the sending rate to the receiving rate in a step-wise manner. This step-wise reduction aims to avoid any overreactions to ephemeral variations in capacity. We provide further details in Section 3.4.

3.2 Collection of the Signals

Setting the Pulling Rate. The pulling rate is calculated at the sender side. It is an estimation of the available bandwidth and serves as a signal guiding the sender in setting its rate increase. ETC places more emphasis on the consistency of the pulling rate instead of the measurement accuracy. We next explain how the pulling rate is set.

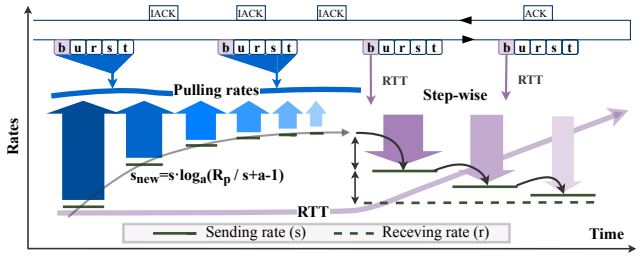


Figure 1: The structure of information collection and rate updates in ETC.

When a flow is first established or a competing flow finishes, the sender has no knowledge of available bandwidth. Hence, we refer to this first stage as the *bandwidth detection stage*. In this stage, ETC directly paces data in micro-bursts to obtain the pulling rate for guidance (in Section 4), which mitigates the disruptions to the network (*e.g.*, changing the sending mode or sending additional probe packets). The pulling rate is set by the ETC sender as the instantaneous receipt rate of the acknowledged micro-bursts. Intuitively, the arrival interval of the packets in each micro-burst is always determined by the last bottleneck it passes (detailed analysis in Appendix F). Therefore, with the collection of many measurement samples, flows that pass through the same last bottleneck should obtain approximately the same estimate of the available bandwidth. Indeed, our analysis in Section 5.1 proves that the measurement results are larger than or equal to the available bandwidth, hence, they can act as the pulling rate to pull the sending rate.

Once the sender does not observe a significant increase (less than $1.1\times$ or even no increase) in the corresponding receiving rate of the past 8 RTTs [1] after a sending rate increase, we assume that the sender has reached the approximate available bandwidth, where only small adjustments are needed for better convergence and fairness. We refer to this as the *stable convergence stage*. Thus, the ETC receiver uses a more conservative method to calculate the pulling rate: $R_p = min(k \times r_{max},\ R_p)$, in which r_{max} is maintained as the maximum receiving rate. We use the parameter k to leave room for exploiting spare bandwidth when a flow ends. When the receiving rate exhibits a sharp increase again (more than $1.1\times$), ETC exits the stable convergence stage to consume the spare bandwidth. Note that r_{max} will only be updated when the rate increases by more than $1.1\times$. When the receiving rate is far from r_{max}, we consider that a new flow joins in and the available bandwidth reduces, therefore r_{max} is sampled again.

Accurate RTT Sampling. RTT is widely adopted as a congestion signal. However, even without considering the statistical error and the shaping of the packets by the bottom layers, the RTT struggles to reflect the actual state of the network due to the bursty nature of TCP. The reason is that bursts may

This is the average receipt rate within a past time window. The time window is typically six to ten RTTs [1] and ETC adopts a time window of eight RTTs.

cause extremely short-lived RTT increases. Regarding this a congestion signal usually damages throughput.

Taking this fact into consideration, we believe that the RTT of the first packet in each micro-burst most accurately reflects the exact network state.Thus, we only regard the RTT of the first packet in each micro-burst as an effective sample. Our statistical results in Appendix E show that the first packet always shows a smaller delay than the other packets in the same micro-burst. This suggests that the first packet better reflects network congestion without being affected by micro-bursts. Note, ETC also uses incremental and unique packet numbers to avoid the effect of retransmission ambiguity on RTT measurements.

3.3 Rate Acceleration

We next explain how ETC exploits the measured pulling rate to accelerate the sending rate in detail. The rate update process is shown in lines 12 to 19 in Algorithm 1. Put simply, when the RTT decreases or eight RTTs (by default) have passed from the last rate increase, ETC raises the sending rate. The increase is calculated as a function of the pulling rate and the sending rate (lines 13-15). The periodical rate increase ensures ETC will not suffer from starvation when competing with more aggressive buffer-filling flows.

To calculate the update, the sender first models the distance between the pulling rate and the current sending rate as a ratio. We use the ratio as it alleviates the impact of any measurement error. Then we link the extent of the rate increase with the distance. Much like an elastic band, the further away from the pulling rate, the faster the sending rate increases. To achieve this, ETC employs a concave formula of the distance, $f(R_p, s)$, which takes the pulling rate as the maximum. The sending rate is promoted in a multiplicative manner: $s = f(R_p, s) \times s$, and the formula should meet the following requirements:

- When the sending rate equals the pulling rate, the sending rate should remain static.
- The sending rate increases should diminish the closer the flow gets to the pulling rate.
- After each adjustment, the difference (inequality) between the sending rates (or the receiving rate) between the flows should decrease.

The first condition guarantees that the sending rate will not exceed the pulling rate. The second condition guarantees that the sending rate increases in a safe way. That is, the sending rate should not spike significantly beyond the available bandwidth (as this would induce congestion). The third condition ensures fairness among flows. To meet these requirements, ETC employs a log function as follows:

$$f(R_p, s) = log_\alpha(R_p/s + \alpha - 1),\ \alpha > 1 \quad \text{s.t.} \quad f(R_p, R_p) = 1 \tag{1}$$

Algorithm 1: ETC Sender Algorithm
t_{last} : the time of the last packet sending.
t_{update} : the time of the last rate update.
s : current sending rate. r : current receiving rate.

```
1  function Send()
2      interval ← now − t_last
3      pending ← s · interval + remain
4      bytes_in_burst ← packet_size · N
5      if pending ≥ bytes_in_burst then
6          send N packets
7          remain ← pending − bytes_in_burst
8          t_last ← now
9      else if now − t_last ≥ 10ms then
10         send pending bytes
11         t_last ← now
12 function RateUpdate()
13     if Δ_RTT < 0 || now − t_update ≥ 8RTTs then
14         s ← s · Eq. 1
15         t_update ← now
16     else if Δ_RTT ≥ threshold then
17         s ← (s + r)/2
18         draining the queue
19         t_update ← now
20 procedure NewACK(ack)
21     update R_p
22     update r
23     if now − t_update ≥ RTT then
24         RateUpdate()
25     Send()
26 procedure TimerExpire()
27     Send()
```

According to our experience, e is a suitable choice for α. We set this as the default parameter in our later evaluation. Note, the selection of α influences the competitiveness of a flow: a smaller base brings a more aggressive flow (the effect of the base on performance is shown in Appendix A.5). The convergence and fairness of flows with this formula are proved in Section 5.3.

3.4 Rate Deceleration with Queue Draining

We next explain how ETC decelerates the rate when the RTT increase exceeds a threshold (lines 16-18). Here, we assume the presence of buffer bloat, indicating the need to drain the router queues. To achieve this, ETC performs two steps:

1. We reduce the sending rate to $r + (s - r)/2$. Theoretically, when the sum of flows' rates exceeds the link capacity, if each flow sets its sending rate to its receiving rate, the queue at the bottleneck will no longer accumulate. How-

ever, due to ACK compression and delays at end-hosts, the measured receiving rate tends to be smaller than the actual value. Hence, we reduce the sending rate to the receiving rate in a step-wise manner. Specifically, the rate difference between s and r is halved at each step to compensate for the measurement imperfections.

2. We update the estimated BDP with the product of the new sending rate and the minimum delay. ETC then checks whether the inflight packets have exceeded one recalculated BDP; if so, it flushes the queue of all remaining packets immediately. Specifically, we flush the accumulated bytes with a sub-sending rate ($\eta \cdot s,\ 0 < \eta < 1$). By doing this, the routers on the path drain their buffers. We estimate the time for draining the queue as follows:

$$t_d = \frac{inflights - s \cdot RTT_{min}}{(1-\eta) \cdot s} \tag{2}$$

We choose to use the sub-sending rate, rather than stopping transmission, because the sender still needs ACKs to trigger data transmission (details in Section 4). After consuming the queue, we resume the sending rate and the rate remains unchanged for at least an RTT (as we can only observe the influence of the current rate after the rate has been kept for an RTT).

Note, the step-wise rate decrease prevents ETC from overreacting. Besides, draining the accumulated packets in the buffer with a low rate can compensate for the imperfections of pacing and ensure continuous ACK response. Although ETC responds to the gradient of the RTT rather than the absolute value, its packet-draining stages ensure that ETC can always converge to both proper rate and minimum latency, without complex parameter adjustment.

4 Transmission in micro-bursts

The previous section has shown how ETC sets the sending rate. We next explain how ETC paces the packets within each micro-burst using the allocated sending rate. This is not trivial because it puts strict requirements on the granularity of the pacing design. Therefore, based on our observation that the arrival intervals of the ACKs are shorter than a commodity system's timer granularity, we propose a novel ACK-clocking pacing mechanism.

4.1 Necessity and Obstacles

Many researchers have observed that TCP's window-based congestion control mechanisms can lead to bursty traffic on modern high-speed networks [25–27]. These bursts bring about buffer bloat and even overflow at intermediate routers. To overcome the problems faced by traffic bursts, *pacing* has long been shipped in Linux and data center networks [26, 28], most of which are dependent on system timers.

ACK Seq	Time(s)-12Mbps	ACK Seq	Time(s)-12Mbps	ACK Seq	Time(s)-100Mbps	ACK Seq	Time(s)-100Mbps
279	5.100818	283	5.105269	279	7.686917	283	7.687006
280	5.101901	284	5.106348	280	7.686937	284	7.687036
281	5.102996	285	5.107473	281	7.686959	285	7.687071
282	5.104066	286	5.108532	282	7.686980	286	7.687098

Table 1: Cut off of the arrival times of ACKs under 12Mbps and 100Mbps links. The complete ACK arrival intervals distribution is shown in Appendix D.

ETC treats *micro-bursts* as the pacing unit. Here, a micro-burst refers to a train of at least two packets. Due to this, the granularity of the system timer should be less than the sending time of two packets to ensure that we can arbitrarily set the micro-burst size. In fast networks, this sending time is too short, *e.g.*, in theory, packets should be sent at $60\mu s$ intervals in the case of 200 $Mbps$ bandwidth and a $1,500$ bytes packet size. However, the sending time is more than an order of magnitude more precise compared with the operating system scheduler's ms order (*e.g.*, $10ms$ [29] and $1ms$ respectively before and after Linux 5.3.0).

Practitioners have come up with a number of solutions to compensate for the inaccuracy of the system timer. Except for those equipped with additional hardware [30, 31], current software-based high-precision pacing methods include timer interrupt-based [28] and gap packet-based ones [32, 33]. Yet these all incur high CPU overhead (see Section 7). Apart from the implementation challenges, pacing that only relies on the system timer triggers a system call for each packet transmission, leading to heavy CPU overhead [26]. Further, to send fixed-size micro-bursts, the timer needs to be dynamically adjusted according to the real-time sending rate ($bytes = rate * interval$) or, alternatively, always be set to the highest precision, which brings further overheads. Thus, ETC proposes a novel approach, which avoids the above limitations.

4.2 ETC Pacing with ACK-Clocking

ETC pacing relies on a simple observation: *the arrival times of the ACKs are more fine-grained than the system's timer and can be utilized for improving pacing precision.* To highlight this, we run flows between Beijing and Shenzhen via $12Mbps$ and $100Mbps$ links (about 37ms delay), and record the arrival times of the ACKs during the transmission. The packets are paced with the $10ms$ system timer and the receiver sends back an ACK for every packet. Table 1 lists the arrival times for a random subset of ACK packets. Although the ACKs' arrival interval is not fixed, the table shows that the interval is more fine-grained than the inherent ms-level timer in the operating system. The average interval of the $12Mbps$ link is around $100us$, and that of the $100Mbps$ link is around $10\mu s$. Yet, the maximum fidelity of the timer in Linux is just $1ms$.

This observation inspires us to exploit ACK-clocking to enhance the accuracy of pacing. Lines 1-11 in Algorithm 1 show how ETC uses micro-bursts (rather than individual packets) as the pacing unit. The sender checks whether a micro-burst should be sent at every ACK arrival. It calculates the pending bytes according to the time interval from the last sending time and the current sending rate (line 3). If the pending bytes outweigh the size of a micro-burst, a micro-burst is sent and the remaining unsent bytes are recorded (line 6-7). Otherwise, the pending bytes will wait for the next ACK arrival. However, ACKs do not arrive in a uniform way because their spaces are altered if they are blocked at the receiver buffer or router queues. Hence, the idle bandwidth will be wasted if a sender keeps waiting for the ACK's arrival. To address this, we keep a timer with 10*ms* precision that all current systems can support. The timer expiration triggers the Send() function. If the pending bytes have been waiting for 10*ms*, these bytes are sent even if they cannot form a micro-burst (line 9-11).

ETC integrates the processing of data sending into the processing of ACKs, keeping only the timers that current system can support. This improves pacing accuracy while avoiding much additional CPU overhead. In addition, since the number of ACKs always corresponds to the number of the data packets, we avoid dynamic adjustment of the timer accuracy according to the network environment.

4.3 Reducing Feedback Frequency

Generally, ACKs take up about 4%-5% of the link capacity if the receiver sends back an ACK for each packet. This bandwidth consumption is enlarged in wireless networks because of the medium access overhead [3, 34, 35]. Intuitively, we can alleviate the interference of excessive ACKs by applying byte-counting ACKs or periodic ACKs. However, ETC relies on ACKs to trigger data transmission and congestion perception. This means such techniques would not work for ETC. Thus, we propose an adaptive feedback mechanism.

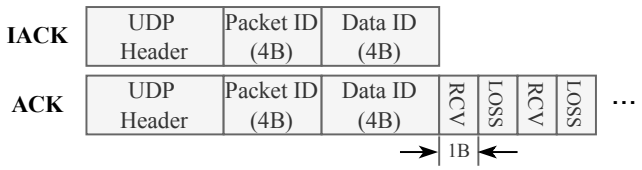


Figure 2: The structure of ACK and IACK.

To reduce the bandwidth consumption brought by ACKs while ensuring timely synchronization, ETC introduces two feedback packets: a lightweight IACK and a standard ACK with detailed information.

Figure 2 shows the structure of ACK and IACK packets. IACKs carry a small amount of data: the packet ID and data ID of the latest and consecutively received packet. Note, the packet ID and data ID are used to let the sender clean the acknowledged packets in time. IACKs play the role of triggering the data sending and data cleaning for the sender. Considering that the sender sends data only when pending packets exceed the micro-burst size, the frequency of IACK feedback can be reduced moderately according to the size of the micro-burst. In cases where the feedback frequency of IACK is deemed insufficient to enable the calculation of the pulling rate (*e.g.*, only one IACK for a micro-burst), we opt to set the pulling rate as $1.25\times$ of the current max receiving rate. Note that this operation has a certain impact on the fairness performance. Our evaluation results (in Appendix A.6) show that responding to every eight packets with an IACK effectively improves the throughput in a wireless network (we always reply to the first packet in each micro-burst with an IACK for RTT collection), without damaging the accuracy of pacing.

However, IACKs cannot convey packet loss information in a timely fashion because they do not contain sufficient information in the header. Hence, if the receiver observes packet loss, it sends an ACK packet instead. The ACK carries the latest consecutively received data-id and packet-id. Then the number of consecutively received packets and lost packets are filled in turn as the loss information. Note that this optimization aims to minimize system overhead, and the default ACK feedback mechanism within the kernel can also accommodate ETC's pacing mechanism.

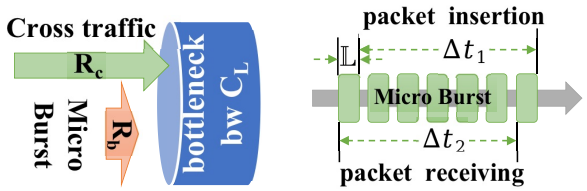
5 Properties of ETC

Our design relies on three core properties. Thus, in this section, we demonstrate these important properties that contribute to the performance of ETC: (*i*) *P1:* The pulling rate is always greater than or equal to the available bandwidth (§5.1). This is vital to prove as it enables the pulling rate to serve as the guide in ETC rate acceleration; (*ii*) *P2:* ETC always seizes spare bandwidth in a safe and fast way (§5.2); (*iii*) *P3:* Competing ETC flows always move towards the fair equilibrium in every rate adjustment (§5.3).

5.1 Pulling Rate $\in$ [ABW, Capacity]

The pulling rate is an important signal in ETC's rate control. To ensure that the pulling rate can guide senders, it is vital that it is always (*i*) greater or equal than the available bandwidth (to probe the capacity); and (*ii*) less than the upper link capacity (to avoid introducing severe congestion).

To prove that the pulling rate fulfills these requirements we next model the instantaneous receipt rate of a micro-burst consisting of N packets and give mathematical proof. But we first start with a simple example. Figure 3(a) shows an example high-speed micro-burst competing with the stable cross traffic for the bottleneck bandwidth C_L. The green arrow represents the cross traffic with a constant rate R_c, and the orange arrow shows the short but quick packet micro-burst at the rate R_b. Intuitively, if $R_b \geq (C_L - R_c)$, the link becomes the bottleneck, otherwise the end-host is the bottleneck. Even



(a) Cross traffic (b) Micro-burst model

Figure 3: Measure the available bandwidth.

if the packets are sent at line rate, according to the principle of fair competition, the measurement result could still be disturbed by background traffic.

Figure 3(b) shows the model of utilizing micro-burst to estimate the available bandwidth. From the arrival of the first byte in, the cross traffic can jump the micro-burst until the last packet arrives (Δt_1). When calculating the receiving rate at the end-hosts, we record the time interval from the arrival of the first packet until the last byte arrives (Δt_2), considering that $N-1$ packets are received during this period.

To express the above in a mathematical form, it costs:

$$\Delta t_1 = \frac{(N-1)*L}{R_b} \tag{3}$$

for the arrival of $N-1$ packets in a single packet train, then the expectation of bytes inserted during this period is $R_c * \Delta t_1$, the Δt_2 can be written as:

$$\Delta t_2 = \frac{(N-1)*L + R_c * \Delta t_1}{C_L} \tag{4}$$

Therefore, the measured pulling rate can be described as:

$$R_p = \frac{(N-1)\cdot L}{\Delta t_2} = C_L/\{\frac{R_c}{R_b}+1\} \tag{5}$$

With this equation, we can prove:

(*i*) *The pulling rate is always greater than or equal to the available bandwidth:*

$$\begin{aligned} R_p - (C_L - R_c) &= C_L/\{\frac{R_c}{R_b}+1\} - (C_L - R_c) \\ &= \frac{R_c^2 + R_c\cdot(R_b - C_L)}{R_b + R_c} > 0 \end{aligned} \tag{6}$$

(*ii*) *The pulling rate is less than or equal to the link capacity:*

$$C_L - R_p = \frac{R_c}{R_c + R_b}\cdot C_L > 0 \tag{7}$$

(*iii*) *The pulling rate is positively correlated with C_L (with $C_L/2$ as the lower bound.)* Equation 5 can be further expressed as:

$$R_p = C_L/\{\frac{R_c}{C_L}\cdot\frac{C_L}{R_b}+1\} \tag{8}$$

$\frac{R_c}{C_L}$ and $\frac{C_L}{R_b}$ are both in the range of (0,1), R_p approaches the lower bound when there is almost no available bandwidth.

To summarize, the above shows that ETC's measured pulling rates will always be greater or equal to the available bandwidth, yet below the link capacity. Note, despite these properties, ETC ensures that sending rates do not excessively exceed the available bandwidth by: (*i*) continuously observing the receiving rate and delay; (*ii*) deploying a concave function to guide rate increase.

5.2 Faster & Safer Start-up than Slow Start

In the start-up stage, ETC senders replace the traditional slow start and, instead, increase the sending rate according to Equation 1. In this subsection, we prove that, compared with slow start, an ETC sender can seize the available bandwidth in a faster and safer way.

Faster. To show that ETC takes less time to achieve higher bandwidth utilization, we employ a simple single-flow mathematical model. Since slow start always (by definition) exceeds the link capacity, and ETC regards the link capacity as the upper bound, we analyze the time required to reach a specific rate ($k\%\cdot C_L$, k is the link utilization). For the slow start, it requires $N = log_2(k\%\cdot C_L)$ RTTs as $2^N = k\%\cdot C_L$. For ETC, the time required to reach high bandwidth utilization is:

$$\prod_{n=1}^{N} f(C_L,\ r_n) = k\%\cdot C_L \tag{9}$$

for which it is hard to derive a neat analytical solution. Therefore, we conduct numerical analysis.

We denote the flow rate as $k\%\cdot C_L$ and make a reasonable assumption that k starts at 0.1. According to Equation 1, we find that when $k \approx 20$, the multiplier of ETC and slow start is equal, that is: $f(C_L, 20\%\cdot C_L) \approx 2$. With Equation 9, we observe that ETC takes five RTTs for k to increase from 0.1 to 20. At this point, ETC achieves a rate approximately **17** times that of slow start. After this stage, the multiplier of ETC becomes smaller than that of slow start. However, slow start requires more than **9** RTTs to be on par with ETC's rate ($\prod_{n=1}^{9} f(C_L,\ r_n) \approx 2^{9.7}$), where k is beyond 85%. From 0 to 9 RTTs, the amount of data transmitted by ETC is **4** times that of slow start ($\sum_{n=1}^{9} f(C_L, r_n) \approx 4\times\sum_{n=1}^{9} 2^n$). This is because ETC increases the rate in a concave manner while, in contrast, the slow start increases the rate in a convex way. The experimental results regarding different initial values of k are shown in Appendix A.7.

Safer. We next show how ETC is *safer* than traditional congestion control algorithms, *i.e.* it rarely overestimates bandwidth. Recall that traditional algorithms employ an exponentially increasing *cwnd*, *i.e.*, the sending rate in n^{th} RTT (s_n) will be twice compared to $(n-1)^{th}$, and the window growth follows $s = 2^n$. In contrast, ETC increases s with a log multiplier (see Eq.1). This indicates that the rate accords with $s_n = s_{n-1}\cdot f(C_L, r_{n-1})$. Since $r < C_L$, it is a concave function within the range $[0, C_L]$, and the rate increment decreases monotonically. In short, the slow start mechanism gains increasingly *cwnd* increments along with RTT. In contrast, ETC adopts gradually converging *cwnd* increments to

avoid capacity overflow. This alleviation reduces the chance of overestimating capacity and triggering congestion.

Under the above settings, *ETC approaches the available bandwidth in a safe manner:* After nine RTTs, ETC has already lifted the utilization beyond 90%. In the next RTT, ETC will slowly converge to the C_L with only a 1.1x amplification. However, the slow start will still double the sending rate even if the link has been fully occupied, which thus seriously oversteps the C_L and incurs severe congestion.

5.3 Fair Convergence

A key goal of ETC is to enable fair convergence across competing flows. We define the convergence and fairness on an ideal steady path (*i.e.*, no flow joins or leaves) as follows.

Definition 1. *A flow **converges** if, there is a time T after which the sending rate is always bounded in the range $[R_{min}(C_L), R_{max}(C_L)]$ (the converged range is related to the link capacity C_L).*

Definition 2. *Consider two flows f_i and f_j starting from arbitrary initial conditions. An algorithm is **fair** if each rate adjustment reduces the difference between the receiving rate r_i and r_j (denoted as $D_{i,j}$) until $D_{i,j} \leq D$, a preset threshold.*

Theorem 1. *With Equation 1, ETC is a **fair** CC algorithm and ETC flows always **converge**.*

Proof. Based on the above definitions, we first analyze whether we can adapt the sending rate of each flow to a fair share at once. Assuming that there are only two flows f_1 and f_2 in the link, where the sending rates are denoted as s_1 and s_2, and the receiving rates are r_1 and r_2, respectively. Under the principle of fair competition, we have:

$$\frac{s_1}{s_2} = \frac{r_1}{r_2} \tag{10}$$

If we multiply s_1 and s_2 by a coefficient k_1 and k_2 so that $s_1 = s_2$, k_1 and k_2, we obtain the following relationship:

$$\frac{k_1}{k_2} = \frac{s_2}{s_1} = \frac{r_2}{r_1} = \frac{C_L - r_1}{C_L - r_2} \tag{11}$$

Consequently, if the flows have already reached the bottleneck bandwidth and calculated their receiving rate, it is feasible for two flows to share the bandwidth fairly after a single tune-up. However, if we expand this scenario to K flows, Equation 11 is further enriched to:

$$\forall i, j \in \mathcal{K}, \quad r_i = \frac{s_i}{\sum_{k \in K} s_k} \cdot C_L, \quad \frac{k_i}{k_j} = \frac{r_j}{r_i} \tag{12}$$

Since this condition requires the receiving rate of all the other flows, the limited information mastered by each flow hinders the realization of 'one-step fairness'. Therefore, we must realize fairness among flows through multiple adjustment iterations. In each adjustment, we multiply the previous sending rate s_i by a function of r_i: $f(r_i)$, $(f(r_i) \geq 1)$. $f(r_i)$ should meet the condition that the absolute difference between the receiving rate of any two flows decreases after adjustment, that is:

$\forall i, j \in K, r_i > r_j$:

$$\left|\frac{f(r_i)s_i - f(r_j)s_j}{\sum_{k \in \mathcal{K}} f(r_k)s_k}\right| \leq \left|\frac{s_i - s_j}{\sum_{k \in K} s_k}\right| \tag{13}$$

Since $f(r_k) >= 1$, we have $\sum_{k \in K} f(r_k)s_k \geq \sum_{k \in K} s_k$, then:

$$\left|\frac{f(r_i)s_i - f(r_j)s_j}{\sum_{k \in \mathcal{K}} f(r_k)s_k}\right| \leq \left|\frac{f(r_i)s_i - f(r_j)s_j}{\sum_{k \in K} s_k}\right| \tag{14}$$

To simplify Equation 13, we propose a more stringent condition based on it:

$$\begin{aligned} &\left|\frac{f(r_i)s_i - f(r_j)s_j}{\sum_{k \in K} s_k}\right| \leq \left|\frac{s_i - s_j}{\sum_{k \in K} s_k}\right| \\ i.e., \quad &|f(r_i)s_i - f(r_j)s_j| \leq |s_i - s_j| \end{aligned} \tag{15}$$

This equation holds only when $s_i = s_j$ $(r_i = r_j)$.

Let s_i^n and r_i^n represent the sending and receiving rate at the n_{th} iteration, Equation 15 can be further written as:

$$|f(r_i^n)s_i^n - f(r_j^n)s_j^n| \leq |f(r_i^{n-1})s_i^{n-1} - f(r_j^{n-1})s_j^{n-1}| \tag{16}$$

Since the equation holds only when $s_i = s_j$, we propose the limits as follows:

$$\begin{aligned} \lim_{n \to \infty} s_i^n - s_j^n &= 0 \\ \lim_{n \to \infty} r_i^n &= \frac{C_L}{K} \end{aligned} \tag{17}$$

Above all, fairness and convergence can be both achieved through a rate increase function that matches the condition in Equation 13. Bringing Equation 13 into Equation 15, the conditions are listed as follows, *i.e.*, if $f(r_i)$ satisfies the following conditions, the flows guided by $f(r_i)$ finally converge and obtain a fair bandwidth share:

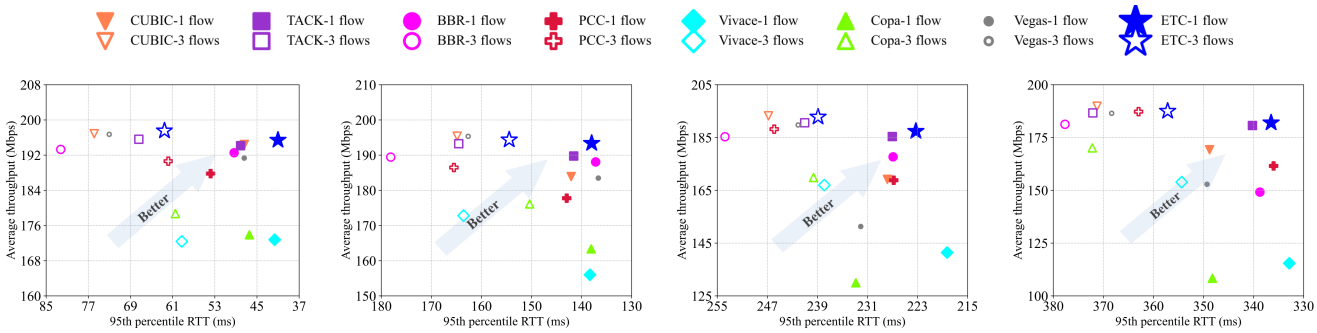
$$\begin{aligned} f(r_i)r_i - r_i &\leq f(r_j)r_j - r_j \\ f(r_i)r_i + r_i &\geq f(r_j)r_i + r_j \end{aligned} \tag{18}$$

Equation 1 satisfies the above two conditions, confirming that ETC shows strong convergence and fairness. Note that Equation 15 is a sufficient and unnecessary condition for fairness and convergence. □

6 Evaluation

6.1 Evaluation Methodology

Implementation. We implement ETC in C++. It is developed as a user-space transport with UDP as a substrate. ETC implements reliability and rate control modules in user space and interacts with the kernel through the *sendmmsg* and *recvmsg* functions (the specific function depends on different platforms). The inherent pacing $10ms$ timer is realized through the *libevent* library. We use this user-space implementation



(a) BJ to SZ (b) BJ to BOM (c) BJ to VA (d) BJ to SAO

Figure 4: The performance of CC algorithms on real links. The links are sorted according to the average latency from low to high. ETC achieves almost the lowest latency and the highest throughput among all the methods.

to test ETC in both emulated and real network paths. *ETC is implemented in the form of SDK and has served as the transmission protocol on Tencent XP2P (with millions of users) for more than a year.* It supports multiple platforms including Windows, macOS, Android, iOS, and OpenWrt (for IoT).

Setup. We compare the performance of ETC with the Linux implementations of several state-of-the-art schemes: CUBIC [9], BBR [1], Vegas [12] (the related parameters are by default), and user-space implementations: PCC [4], PCC Vivace [5], Copa [2], and TACK [3]. Among these algorithms, CUBIC serves as a representative of loss-based approaches; Vegas and Copa are delay-based algorithms; BBR, PCC, and Vivace are representative of learning-based solutions; and TACK makes optimizations on the transmission feedback. We use Pantheon [36], an independent platform that serves as a "training ground" for research on congestion control. We deploy Pantheon in Cloud servers distributed in Beijing (BJ), Shenzhen (SZ), Bombay (BOM), Sao Paulo (SAO), and Virginia (VA). We always run flows from Beijing to other cities (the base RTT of the links is shown in Appendix B). The bandwidth between every two nodes is limited to 200$Mbps$. The situations that are difficult to reproduce in real Internet paths (*e.g.*, links with high random loss) are simulated with the Pantheon locally. We further evaluate its performance on a video playback use case.

Parameters. In the server tests, considering that the throughput of TCP variants is limited by the sliding window, we dynamically set the TCP rmem/wmem to 2×BDP. The receive buffer of UDP is 2MB. The buffer size in the simulation scenario is set to 1×BDP. These settings ensure that the performance of the CC algorithms is not limited by the buffer size. Referring to the studies of ABW measurement, we set the size of micro-burst to 8. The base of the *log* function is e. The initial sending rate is 10 packets per RTT (by default after Linux 3.0.0) and η is set to 0.3. The traces used in emulations are generated according to Mahimahi's [37] rules.

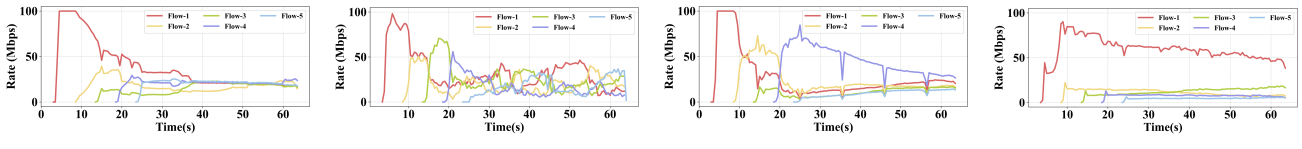
We will include the GitHub URL of the traces upon publication.

6.2 Throughput & Delay

To evaluate the throughput and delay achieved by ETC, we test the algorithms across the five deployed real-world nodes. Although the performance of single flow has been widely inspected, little attention has been paid to the multi-flow performance. Therefore, we run each algorithm with both a single flow and three flows 30 times to check their performance under multi-flow scenarios. Note, we evaluate asynchronous flow arrival in Section 6.3. The experiments run during different time periods and the represented results are the average across all the results. Flows in each experiment start simultaneously and keep running for 30s.

Figure 4 presents the average throughput and 95th one-way delay (OWD) of the evaluated protocols on the real Internet paths. As Figure 4(a) shows, CUBIC, BBR, PCC, Vegas, TACK, and ETC all achieve good throughput performance in the short path from Beijing to Shenzhen, among which ETC has the shortest OWD with equivalent throughput. From all the sub-figures in Figure 4, BBR shows a low OWD with a single flow, yet BBR always attains the longest OWD in multi-flow scenarios due to its excessive estimation of the available bandwidth. Based on BBR, TACK adds the perception of the buffer, which optimizes the delay performance of BBR. Despite the poor competitiveness of Vegas, a single Vegas flow shows good performance in both delay and throughput, yet it also shows obvious OWD increment as the flow number increases. Compared with these algorithms, Copa and Vivace attain 20% throughput waste in exchange for low latency. As the OWD of the paths grows (from subfigure 4(a) to 4(d)), all the protocols show varying degrees of throughput decline. We observe that ETC and TACK still maintain high link utilization on long links though, with a 90% utilization on a 400ms link. As an aggressive protocol, PCC's throughput loss comes from its long decision cycle (four RTTs). Vivace and Copa are sensitive to OWD measurement errors and their throughput reduces to about only half of the link capacity.

Our statistics also demonstrate that CUBIC, BBR, and TACK show comparatively higher loss rates. Specifically,



(a) CUBIC-total TPT: 98.63Mbps (b) TACK-total TPT: 98.00Mbps (c) BBR-total TPT: 97.05Mbps (d) PCC-total TPT: 89.44Mbps

(e) Vivace-total TPT: 76.16Mbps (f) Copa-total TPT: 85.28Mbps (g) Vegas-total TPT: 98.47Mbps (h) ETC-total TPT: 98.52Mbps

Figure 5: Variation of the throughput as five flows gradually join the network. ETC outperforms other protocols by a large margin *w.r.t.* bandwidth allocation fairness and convergence speed.

ETC reduces packet loss rates by 55% and 42% compared to CUBIC and BBR, respectively. This is due to ETC's constraint on the pulling rate (as outlined in Section 3.2) and timely queue draining during rate deceleration (in Section 3.4).

To summarize, even on short links, ETC shows 10% and 30% OWD improvement in one-flow or three-flow scenarios compared with BBR, respectively. Compared with Copa and Vivace, ETC shows more than 15% throughput advantage with almost the same OWD. Besides, ETC consistently maintains a low loss rate.

6.3 Fairness & Convergence

To verify the fairness of ETC, we test the bandwidth occupation of each algorithm when starting N flows simultaneously and at fixed intervals (5 seconds). We run the fairness test in the real-world link from Beijing to Shenzhen, with a delay of 37ms. Each flow runs for 60s.

Figure 5 shows how these algorithms perform when five flows gradually join a 100$Mbps$ link with a 5s interval. Each line in the figure represents a flow. We evaluate the performance of each algorithm in terms of convergence speed and the fairness of the convergence. We classify a flow as converged once its rate oscillates with variance less than 5$Mbps$ for at least 2 seconds. As Figure 5(h) shows, ETC converges rapidly after the arrival of the new flows, within about two seconds. After all of the flows arrive at 25 seconds, flows oscillate around the convergence rate, and the difference in the flow rates gradually decreases. In the whole convergence process of ETC, there is almost no sharp oscillation. Protocols such as CUBIC (Figure 5(a)), Vegas (Figure 5(g)) and BBR (Figure 5(c)) also maintain stable and fair bandwidth share after convergence. However, they take a much longer period to converge: CUBIC and Vegas take about 15 seconds, and BBR takes over 30 seconds. Vivace (Figure 5(e)) and Copa (Figure 5(f)) converge to a relatively fair bandwidth share but the rate fluctuation is obvious. TACK (Figure 5(b)) and PCC (Figure 5(d)) perform worst in convergence: TACK never converges and PCC is far from fairness.

6.4 Coexistence with Loss Oriented Scheme

It is important that any newly deployed algorithm is friendly to existing schemes. Given that (loss-based) CUBIC is still the current mainstream CC method, we test the performance of different methods when they compete with CUBIC. We run two flows simultaneously in an emulated link with 100$Mbps$ capacity and 25ms RTT, one with CUBIC and the other with one of the comparison algorithms (CUBIC, BBR, TACK, Vivace, PCC, Copa, ETC) for 60s. Considering that the performance of CUBIC is closely related to the bottleneck buffer size, we also run experiments under multiple buffer sizes: 0.5× to 3×BDP.

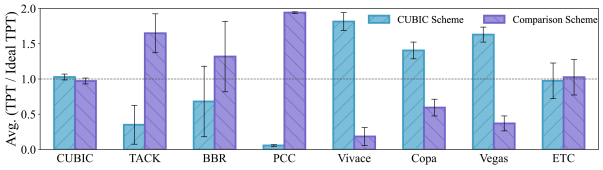


Figure 6: Throughput of the flows under comparison with CUBIC. ETC shows a good throughput without damaging the CUBIC. While PCC, TACK, and BBR greatly compress the throughput of CUBIC.

Figure 6 shows the ratio of the average throughput to the ideal throughput, in which the error bars indicate the ratio under different buffer sizes. We can observe that BBR, PCC, and TACK greatly compress the CUBIC flow, and PCC even obtains a 10× higher throughput than CUBIC. Meanwhile, Copa and Vivace show poor competitiveness under all the buffer sizes. ETC obtains an equivalent performance when competing with CUBIC without damaging its performance.

The competitiveness of ETC comes from the periodic rate acceleration and the fact that it starts with a relatively high initial rate. Further, the additional queue-draining mechanism keeps it from being overly aggressive. When ETC coexists

with CUBIC, the delay is kept at a high level since CUBIC continuously fills the queue. With the update of RTT, the min RTT measured by ETC increases accordingly and the stalling time shortens, ensuring that ETC does not fail to exploit too much bandwidth.

6.5 Video Transmission

To test the performance of ETC in a real application, we integrate ETC in a video player to test its performance against the congestion control algorithms embedded in the Linux system (Reno, CUBIC, Vegas, BBR). In this video player, all of the lost packets are recovered. Figure 7 shows the rebuffer rates of the video flows at different bandwidths and packet loss rates. The bit rate of the video flow is fluctuating around $12Mbps$. With current algorithms, it is not possible to achieve 100% link utilization. That is, a $12Mbps$ link cannot support a video flow of this magnitude. Therefore, we assign the link with various bandwidth levels that can only just support such a video flow, plotted on the X-axis. We do this to observe the influence of the bandwidth/loss rate changes (the bandwidth and loss rates are set by traffic control (TC) in Linux).

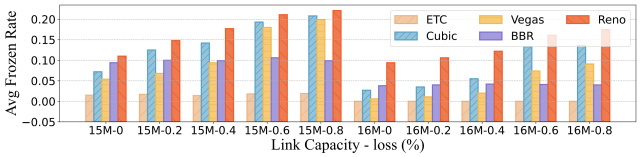


Figure 7: The rebuffer rates of video playbacks under different bandwidths and loss rates.

With a $16Mbps$ link, ETC maintains a zero rebuffer rate even if the loss rates increase. BBR is insensitive to the loss rates and keeps about a 4% rebuffer rate. The other three algorithms show an increasing trend in the rebuffer rate as the loss rate increases. The rebuffer rate of Vegas and Reno approaches 20% when there is a 0.8% loss. When the bandwidth is reduced to $15Mbps$, ETC starts to experience a rebuffer rate of about 2% (almost unaffected by the loss rate). In contrast, the rebuffer rates of other algorithms all exceed 10% as the loss rate increases. This confirms that ETC makes more efficient use of the bandwidth and reduces the rebuffer rate of video playback, especially in the case of limited bandwidth.

6.6 Real Deployment and Other Evaluations

By reporting the data collected from our XP2P SDK (comprising over ten million samples), we demonstrate that ETC reduces the average buffering rate by approximately 11% compared to QUIC. Furthermore, in weak network environments (*e.g.*, when the base RTT of a link exceeds 150ms.), the reduction can reach up to 17%.

We have also conducted experiments to verify the performance of ETC under several other scenarios: (*i*) the performance of competing flows on paths with different RTTs; (*ii*) the performance of flows under different loss rates; (*iii*) the ability for ETC to adapt to bandwidth fluctuations; (*iv*) the fairness among multiple (up to 10) flows. Due to the limit of space, we present these experiment results in Appendix A.

7 Related Work

Delay-sensitive CC Algorithms. Vegas [12], FastTCP [13], and Westwood [38] are representative algorithms based on RTT measurement. They reduce the transmission rate when RTT growth is detected, which performs poorly when competing with buffer-filling schemes. Copa [2] proposes a target rate model and adjusts the *cwnd* in the direction of the obtained target rate. Note, Copa proposes a competitive mode to guarantee its competitiveness.

Online-learning CC Algorithms. BBR [1] tries to learn a better sending rate with the periodical bandwidth probing stage. PCC [39], Vivace [5], and Remy [40] all guide the online actions of the flows via an objective function. With these approaches, the senders continuously observe the connection between their actions and empirically experienced performance, aiming at consistently adopting actions that achieve high performance. These methods bring a heavy burden on the CPU and thus hinder large-scale deployment. Furthermore, our experiment results in A.3 indicate that compared to these algorithms, ETC exhibits superior adaptability to bandwidth fluctuations, while also achieving faster response times.

High-resolution pacing. Practitioners have proposed various solutions to realize high-precision pacing. For hardware-based ones, [30] use the high-resolution timer in the smart network interface controller (NIC), to maintain all TCP connections and schedule packets for each flow. Instead of a per-flow transmission timer, [31] proposes a per-packet transmission timer, which frees the NIC from the upper transport layers. The software-based methods can be divided into timer interrupt-based [28] and gap packet-based ones [32, 33]. The former requires the assistance of a microsecond resolution timer per flow in Gigabit networks and the latter expects the device to transmit both gap and data packets at a wire-rate, bringing unavoidable errors and high overhead. In contrast, ETC paces packets in micro-bursts without requiring any hardware assistance or causing excessive CPU overhead.

Available bandwidth measuring. Bart [41], TOPP [42], Pathload [23], and Pathchirp [22], offer basic solutions for measuring available bandwidth using the packet rate method. Bart [41] samples the available bandwidth of the network path by sending sequences of probe packet pairs at randomized rates. TOPP [42] determines the available bandwidth by sending packet pairs at an increasing rate and analyzing the input and output rates of different packet pairs. Pathload [23] utilizes pre-defined probing rates to transmit packet trains, gradually increasing the rate until the one-way delay shows

an increment. Pathchirp [22] employs 'chirps' of probe packets (which are exponentially spaced) instead of packet trains. This method can assess the available bandwidth with fewer packets. Compared to these algorithms, ETC employs a simple measurement approach that does not interfere with regular data transmission, enabling each flow to quickly obtain a consistent estimate of the available bandwidth.

8 Conclusion

This paper introduces ETC, a congestion control mechanism that provides high throughput, low latency, and good fairness. ETC defines the concept of the pulling rate, which is a pre-congestion consensus signal that works before packets begin to accumulate in the bottleneck buffers. ETC uses the pulling rate to guide the rate acceleration, figuring out dynamic step sizes with a novel elastic principle. Our evaluation results show that ETC achieves superior performance compared with the state-of-the-art algorithms. ETC is also highly practical since it requires no additional hardware or software assistance. Thus, for over a year, it has successfully served as the transmission protocol for our commercial video application, handling millions of users. We believe ETC provides a new perspective on transport protocols for future WANs. Our next line of future work is to explore the efficacy of ETC in wireless scenarios. This work does not raise any ethical issues.

Acknowledgment

This work is supported by the Major Key Project of PCL under grant No. PCL2023A06-4, the National Key Research and Development Program of China under grant No. 2022YFB3105000, and the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989.

References

[1] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14(5):20–53, 2016.

[2] Venkat Arun and Hari Balakrishnan. Copa: Practical delay-based congestion control for the internet. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 329–342, 2018.

[3] Tong Li, Kai Zheng, Ke Xu, Rahul Arvind Jadhav, Tao Xiong, Keith Winstein, and Kun Tan. Tack: Improving wireless transport performance by taming acknowledgments. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 15–30, 2020.

[4] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. {PCC}: Re-architecting congestion control for consistent high performance. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 395–408, 2015.

[5] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. {PCC} vivace:{Online-Learning} congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, 2018.

[6] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, et al. Xlink: Qoe-driven multi-path quic transport in large-scale video services. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 418–432, 2021.

[7] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Analysis and design of the google congestion control for web real-time communication (webrtc). In *Proceedings of the 7th International Conference on Multimedia Systems*, pages 1–12, 2016.

[8] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. Congestion control for web real-time communication. *IEEE/ACM Transactions on Networking*, pages 2629–2642, 2017.

[9] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64–74, 2008.

[10] Sally Floyd, Tom Henderson, Andrei Gurtov, et al. The newreno modification to tcp's fast recovery algorithm. 1999.

[11] Mario Hock, Roland Bless, and Martina Zitterbart. Experimental evaluation of bbr congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pages 1–10, 2017.

[12] Lawrence S Brakmo, Sean W O'Malley, and Larry L Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications architectures, protocols and applications*, pages 24–35, 1994.

[13] Cheng Jin, David X Wei, and Steven H Low. Fast tcp: motivation, architecture, algorithms, performance. In *IEEE INFOCOM 2004*, volume 4, pages 2490–2501. IEEE, 2004.

[14] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 44–58. 2019.

[15] Sándor Molnár, Balázs Sonkoly, and Tuan Anh Trinh. A comprehensive tcp fairness analysis in high speed networks. *Computer Communications*, 32(13-14):1460–1484, 2009.

[16] Guojun Jin and Brian Tierney. Netest: A tool to measure the maximum burst size, available bandwidth and achievable throughput. In *International Conference on Information Technology: Research and Education, 2003. Proceedings. ITRE2003.*, pages 578–582. IEEE, 2003.

[17] Attila Pásztor and Darryl Veitch. The packet size dependence of packet pair like methods. In *IEEE 2002 Tenth IEEE International Workshop on Quality of Service (Cat. No. 02EX564)*, pages 204–213. IEEE, 2002.

[18] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. What do packet dispersion techniques measure? In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 2, pages 905–914. IEEE, 2001.

[19] Robert L Carter and Mark E Crovella. Measuring bottleneck link speed in packet-switched networks. *Performance evaluation*, 27:297–318, 1996.

[20] Kevin Lai and Mary Baker. Measuring bandwidth. In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 1, pages 235–245. IEEE, 1999.

[21] Thomas E Anderson, Andy Collins, Arvind Krishnamurthy, and John Zahorjan. Pcp: Efficient endpoint congestion control. In *NSDI*, 2006.

[22] Vinay Joseph Ribeiro, Rudolf H Riedi, Richard G Baraniuk, Jiri Navratil, and Les Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Passive and active measurement workshop*, 2003.

[23] Manish Jain. Pathload: A measurement tool for end-to-end available bandwidth. In *Proc. of Passive and Active Measurements (PAM) Workshop, Mar. 2002*, 2002.

[24] Sangtae Ha and Injong Rhee. Taming the elephants: New tcp slow start. *Computer Networks*, 55(9):2092–2110, 2011.

[25] Ethan Blanton and Mark Allman. On the impact of bursting on tcp performance. In *International Workshop on Passive and Active Network Measurement*, pages 1–12, 2005.

[26] Amit Aggarwal, Stefan Savage, and Thomas Anderson. Understanding the performance of tcp pacing. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, pages 1157–1165, 2000.

[27] Lixia Zhang, Scott Shenker, and Daivd D Clark. Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic. In *Proceedings of the conference on Communications architecture & protocols*, pages 133–147, 1991.

[28] Salvatore Pontarelli, Giuseppe Bianchi, and Michael Welzl. A programmable hardware calendar for high resolution pacing. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6. IEEE, 2018.

[29] Luca Abeni, Ashvin Goel, Charles Krasic, Jim Snow, and Jonathan Walpole. A measurement-based analysis of the real-time performance of linux. In *Proceedings. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 133–142, 2002.

[30] Makoto Nakamura, M Inaba, and K Hiraki. End-node transmission rate control kind to intermediate routers-towards 10gbps era. *PFLDnet2004, February*, 2004.

[31] Katsushi Kobayashi. Transmission timer approach for rate based pacing tcp with hardware support. *PFLDnet 2006*, 2006.

[32] Hiroyuki Kamezawa, Makoto Nakamura, Junji Tamatsukuri, Nao Aoshima, Mary Inaba, and Kei Hiraki. Inter-layer coordination for parallel tcp streams on long fat pipe networks. In *SC'04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, pages 24–24. IEEE, 2004.

[33] Ryousei Takano, Tomohiro Kudoh, Yuetsu Kodama, Motohiko Matsuda, Hiroshi Tezuka, and Yutaka Ishikawa. Design and evaluation of precise software pacing mechanisms for fast long-distance networks. *PFLDnet 2005*, 2005.

[34] Eugenio Magistretti, Krishna Chintalapudi, Bozidar Radunovic, and Ramachandran Ramjee. Wifi-nano: reclaiming wifi efficiency through 800 ns slots. *ACM/IEEE International Conference on Mobile Computing and Networking*, 2011.

[35] Lynne Salameh, Astrit Zhushi, Mark Handley, Kyle Jamieson, and Brad Karp. Hack: hierarchical acks for efficient wireless medium utilization. *USENIX Annual Technical Conference*, 2014.

[36] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for internet congestion-control research. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 731–743, 2018.

[37] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate {Record-and-Replay} for {HTTP}. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 417–429, 2015.

[38] Claudio Casetti, Mario Gerla, Saverio Mascolo, Medy Y Sanadidi, and Ren Wang. Tcp westwood: end-to-end congestion control for wired/wireless networks. *Wireless Networks*, 8(5):467–479, 2002.

[39] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. {PCC}: Re-architecting congestion control for consistent high performance. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 395–408, 2015.

[40] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. An experimental study of the learnability of congestion control. *ACM SIGCOMM Computer Communication Review*, pages 479–490, 2014.

[41] Svante Ekelin, Martin Nilsson, Erik Hartikainen, Andreas Johnsson, J-E Mangs, Bob Melander, and Mats Bjorkman. Real-time measurement of end-to-end available bandwidth using kalman filtering. In *2006 ieee/ifip network operations and management symposium noms 2006*, pages 73–84. IEEE, 2006.

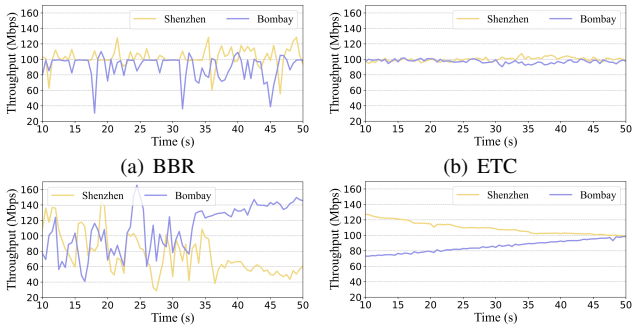
[42] Bob Melander, Mats Bjorkman, and Per Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Globecom'00-IEEE. Global Telecommunications Conference. Conference Record (Cat. No. 00CH37137)*, volume 1, pages 415–420. IEEE, 2000.

[43] Rüdiger Lehmann. 3 σ-rule for outlier detection from the viewpoint of geodetic adjustment. *Journal of Surveying Engineering*, pages 157–165, 2013.

A Supplementary to ETC Performance

A.1 Throughput under Different RTTs

To show the performance of the protocols on links with different delays, we start two flows to SZ and BOM respectively, and limit the throughput of the sender to 200*Mbps*. The base transmission delays are shown in Figure 13. Figure 8 shows the performance of several representative protocols after the slow start at the beginning.



(a) BBR (b) ETC

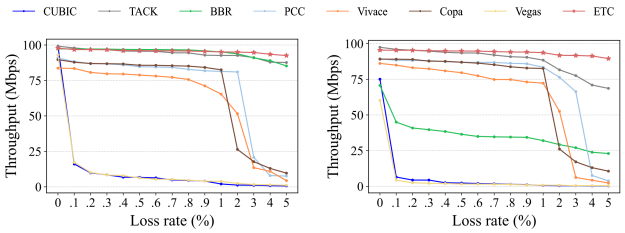
(c) Copa (d) Vegas

Figure 8: Flows in paths with different RTTs. ETC flows obtain almost the same bandwidth share.

For BBR, the flow on the short paths obtains more bandwidth, because BBR enhances the pacing rate every 8 RTTs, which means flows on shorter links are more likely to grab bandwidth (detecting a higher receiving rate). Further, BBR performs RTT probing every 10 seconds if the minimum RTT is not updated. The senders in RTT probing only sends four packets per RTT, which further damages the performance of flows on long paths. Copa experiences a severe throughput degradation, but the flow on the long link always obtains a throughput advantage, because Copa is very sensitive to latency. Once the queue accumulates even several packets, the flow on the short link reacts more quickly and gives up bandwidth. The Vegas flow on the short path increases to the higher bandwidth (due to the slow start) and then converges to the fair position. As Figure 8(d) shows, the additive adjustment brings Vegas an extremely slow convergence and the two flows expense about 60s to fair convergence. ETC responds to RTT gradient, which weakens the relationship between adjustment frequency and RTT, thus ETC performs outstanding fairness on links with different RTTs.

A.2 Resistance to Packet Loss

Algorithms that rely on packet loss as a congestion signal suffers from random or non-congestion packet loss. This is particularly problematic in unreliable wireless networks. To verify that ETC has good resistance to non-congestion packet losses, we test the performance of these algorithms under 0-5% random packet loss. Considering that some algorithms incorporate both delay and packet loss, we set a 100*Mbps* link with 37*ms* or 141*ms* RTT (note, these are the link characteristics of BJ to SZ and BJ to BOM). Ideally, when there is a 5% random packet loss, the throughput of each flow should drop by 5% (rather than tos almost zero).



(a) RTT-37*ms* (b) RTT-141*ms*

Figure 9: The throughput of the protocols with loss rates from 0-5% in links with different RTTs. ETC, TACK, and BBR show good resistance to packet loss.

Figure 9 shows the experimental results. CUBIC and Vegas have almost no robustness when facing packet losses. When the packet loss rate reaches 0.1%, the throughput drops to almost 0. Compared with CUBIC and Vegas, Vivace and PCC have stronger resistance to packet loss. However, when the random loss rate exceeds 3%, their throughput sharply decreases, because the loss rate measurements become more pronounced. BBR, TACK, and ETC all show strong resistance to packet loss. From 0 to 5%, the throughput of ETC dropped by about 5%, and BBR and TACK dropped by about 10%.

A.3 Coping with Bandwidth Fluctuations

In strongly fluctuating links, the performance of the transmission protocols can be evaluated from three perspectives: (*i*) rapid detection and grasp of bandwidth growth for higher link utilization; (*ii*) timely convergence to the available bandwidth for lower latency when there is a capacity reduction; and (*iii*) keeping stable sending rate in each stable period (capacity unchanged).

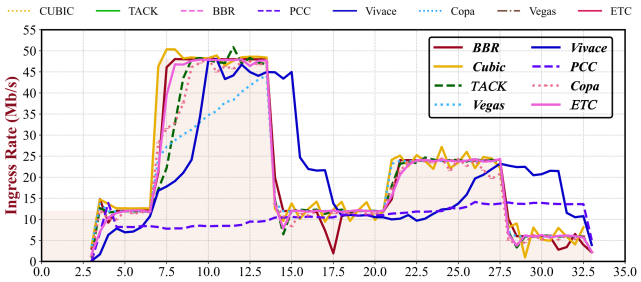


Figure 10: Ingress and egress bandwidth variation in a fluctuated link. The light orange area indicates the real link capacity. The proposed ETC makes full use of the link capacity and avoids congestion in the meantime. Other algorithms fail to achieve high utilization, slow jitter, and the appropriate rate at the same time.

Figure 10 shows the transmission performance of a flow under different protocols which constantly run for 30s. Note that there is no background traffic or competing flows. The link's capacity varies at several time points, and the buffer size is set to 5MB. The background color shows the link capacity, where the bandwidth increases from 12$Mbps$ to 48$Mbps$ at 6s, decreases to 12$Mbps$ at 14s, rises again to 24$Mbps$ at 21s, and finally decreases to 6$Mbps$ at 28s. Ideally, the rate of data arriving at the port and leaving the port should be consistent with the link bandwidth. At this time, the bandwidth can be fully utilized and there is no queue in the link.

Capacity growth: When the flows start, from the left enlarged part in the egress figure, we see that ETC grows the fastest. At about 6s, the link capacity grows, and CUBIC, ETC, and BBR all quickly detect the free bandwidth and fill it up. As the above ingress figure shows, at about 7s, CUBIC exceeds the link bandwidth and continues to grow until the bottleneck buffer spills over, which is determined by the characteristic of the loss-based algorithm. At the same time, neither ETC nor BBR exceeds real link capacity. As for the TACK, Ledbat, and Vivace, they grow slowly and reach full bandwidth after 10s. Among them, TACK exceeds the link capacity for a while at about 11s. PCC is extremely insensitive to bandwidth growth and continues to oscillate near the rate before.

Capacity reduction: When the bandwidth decreases significantly at 14s, ETC, BBR, CUBIC, Ledbat, TACK, and PCC all begin to taper with the link capacity. As the right detailed part in the egress figure shows, ETC starts to reduce the speed at about the right time when the capacity reduces and completes the reduction process soon. The rate reduction of Vivace starts before the link capacity varies, which indicates the reason for the reduction in the queue accumulation caused by the previous excessive transmission rate instead of link fluctuation. As the ingress figure shows, PCC keeps a higher rate and does not converge to the right rate during the 14-20s. The egress figure shows that BBR begins to react at 14s after the link variation (at 13.5s) and quickly completes the deceleration process. However, BBR still operates downward detection operation after reaching 12$Mbps$ (at 15-17.5s).

Stable link: From 15s to 20s, as the ingress figure shows, the link capacity keeps steady at 12$Mbps$. ETC, TACK, and Ledbat fit the link capacity well, while CUBIC and BBR have significant oscillation. Through data analysis, we find CUBIC causing packet loss more frequently in a thinner link, and the measurement accuracy of BBR decreases. The other algorithms send data at a rate slightly lower than the capacity.

Above all, TACK, Ledbat, and PCC respond slowly to the growth of bandwidth. The overall rate adjustment of Vivace (during rate increase or decrease) is lagging. Although CUBIC can occupy the full bandwidth, the ingress rate continues to exceed the link capacity, resulting in data accumulation in the bottleneck. BBR fluctuates seriously when there is a thin link, resulting in bandwidth utilization damage. These two figures both show that ETC makes timely, fast, and appropriate responses to various situations.

A.4 Fairness among Multiple Flows

To verify whether ETC can still keep good fairness when the number of flows increases, we start N flows simultaneously in the real-world link from Beijing to Shenzhen. The base delay is about 37ms. Each flow runs for 60s. We use the Jain Fairness Index to quantify fairness and its expression is as follows (The closer the value is to one, the better the fairness is):

$$F = \left(\sum_{n}^{i=1} \frac{T_i}{O_i}\right)^2 \Big/ \left(n \sum_{n}^{i=1} \left(\frac{T_i}{O_i}\right)^2\right) \tag{19}$$

where T_i represents the throughput of flow i and O_i represents the total throughput of the n flows.

Flow Num.	PCC	BBR	Cubic	Fillp	Copa	ETC	Vivace
3	0.86	0.89	0.89	0.62	0.87	0.93	0.83
4	0.89	0.86	0.89	0.68	0.88	0.96	0.83
5	0.75	0.84	0.84	0.76	0.84	0.94	0.84
6	0.76	0.87	0.88	0.84	0.84	0.90	0.76
7	0.75	0.82	0.86	0.88	0.87	0.89	0.55
8	0.61	0.86	0.86	0.85	0.83	0.91	0.55
9	0.75	0.90	0.89	0.84	0.84	0.91	0.79
10	0.80	0.88	0.82	0.86	0.84	0.88	0.70

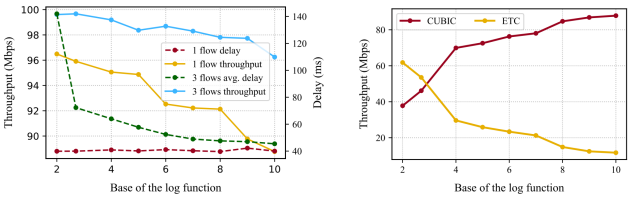
Table 2: Jain Fairness Index of the tested protocols under different flow numbers. The higher the value, the better the performance.

When we start multiple flows at the same time, as Table 2 shows, the Jain Fairness Index of BBR, CUBIC, Ledbat, and ETC are relatively high, which means they achieve good fairness. With TACK, PCC, and Vivace, the bandwidth of each flow oscillates drastically over time and the average bandwidth varies a lot, leading to low Jain indices.

A.5 Influence of the Base Value

We test the influence of different bases on a link with a base RTT of 30ms and max throughput of 100$Mbps$. We run the experiments under both a single flow and three flows. The base is chosen from the set of (2, e, 3, 4, 5, 6, 7, 8, 9, 10). Intuitively, a larger base means that ETC is getting less aggressive and the probability of packet accumulation at the bottleneck decreases. From Figure 11(a), we can observe that as the base increases, the throughput of a single flow decreases significantly (about 10%), while its delay does not change by a noticeable margin. On the contrary, when there are three flows running simultaneously, the total throughput of the three flows does not decrease significantly (only by about 4$Mbps$).

However, the latency is greatly reduced, especially when the base increases to e, and the average latency is reduced by half.



(a) The ETC throughput and latency of one or three flows *w.r.t.* the base of the log function.

(b) The bandwidth allocation for CUBIC and ETC as the base grows.

Figure 11: The influence of the base value (α in the Eq 1) to throughput, delay and competitiveness. The tested link capacity is 100*Mbps*. The base RTT is 30*ms* and the buffer size is 1×BDP.

Figure 11(a) might lead one to hold the view that the larger the base, the better the performance (delay reduction without great throughput decline). In fact, the base also affects the ETC's competitiveness. As shown in Figure 11(b), when compared with CUBIC, the bandwidth allocated to ETC gradually decreases as the base increases. When the base value is e, ETC and CUBIC are basically in fair competition. When the basis value increases further, the competitiveness of ETC decreases sharply. Taking the above observations into consideration, we believe that a base between e and 4 is appropriate in similar environments. The selection of the base can be adjusted according to the specific environment.

A.6 Influence of the IACK Frequency

The number of ACK packets can bring severe medium acquisition overhead in wireless networks (since the wireless network usually employs IEEE 802.11 medium access control (MAC) protocol). Therefore, we conduct experiments under a wireless network to test the influence of feedback frequency on delay and throughput (This hardly influences the throughput in WAN networks). The link capacity is about 52.5*Mbps* (measured with the UDP protocol through iperf) and the base delay is about 18ms.

X packets per IACK	1	2	4	8	16
RTT (ms)	18	19	18.8	19	19.4
Bandwidth (Mbps)	40.55	45.23	47	50.51	51.12

Table 3: The influence of the IACK frequency on flow delay and throughput.

We run a single flow and there is not any background traffic. Table 3 shows the throughput and delay of the flow when we reply to every 1/2/4/7/16 packet(s) with an IACK. We can observe that there is no obvious change in RTT, which means there are almost no accumulated packets in the router buffer and the link capacity is not the key factor that hinders high throughput. As the frequency of IACKs decreases, the throughput gradually increases. When the feedback frequency is reduced from every eight packets to every sixty packets, there is no evident optimization in throughput. Considering the need to guarantee that packets are sent in the form of micro-bursts, we choose to reply with an IACK for every eight packets.

A.7 Influence of the initial k Value

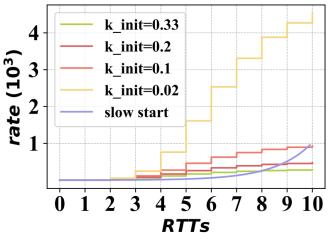


Figure 12: The rate variation of slow start and ETC with different initial values of k. The initial value of the rate (y-axis) is 1.

This section demonstrates the impact of the initial value of k (k_{init}) on the rate variation of ETC (while slow start is not affected by the k_{init}). A smaller k_{init} value corresponds to lower initial bandwidth utilization. From Figure 12, it can be observed that with smaller k_{init}, ETC exhibits a more pronounced advantage over slow start because slow start does not adapt its step changes to reflect the available bandwidth. Even when the k_{init} value increases to 0.33, ETC still demonstrates a rate growth advantage.

B Base Delay of the Tested Links

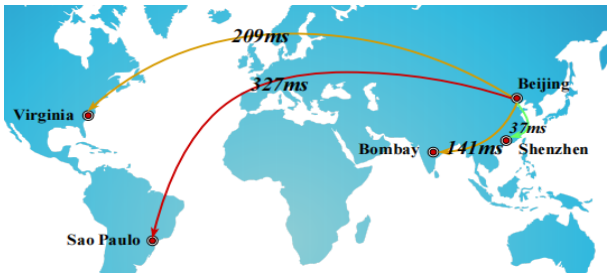


Figure 13: The distribution of the five testing nodes. The average RTT of links is marked on the connection line.

Figure 13 shows the distribution of our test servers and the average delay measured through Ping method. We collect the latency measurements in the morning, noon, afternoon,

and evening through the *Ping* method, then the average latency is regarded as the reference for subsequent performance observations and are marked on the links.

C Convergence of Current CC Algorithms

Based on the opinion that fairness is improved when a rate change reduces the difference (absolute value) between competing flows. We provide a brief analysis of current algorithms' techniques for achieving fair convergence.

Current traditional loss-oriented algorithms, *e.g.*, NewReno [10] and CUBIC [9], update their congestion windows in an **A**ddictive **I**ncrease **M**ultiplicative **D**ecrease (AIMD) manner. NewReno's addictive operation increases the *cwnd* with a single packet per RTT and halves the cwnd upon packet drop occurs. It depends on the multiplicative rate decrease to drive the flows toward fairness since flows with higher rates drop more. CUBIC figures out the addictive step size according to a cubic function, further promoting convergence speed. However, this only holds when all of the flows maintain the same maximum window (the *cwnd* when packet drop occurs). Therefore, CUBIC usually experiences several buffer overflows before CUBIC flows converge.

Next, delay-sensitive algorithms are represented by Vegas [12] and Copa [2] (we consider Copa as a delay-sensitive protocol because it mainly uses the RTT signal). As a **A**ddictive **I**ncrease **A**ddictive **D**ecrease (AIAD) algorithm, Vegas can still move towards fairness because Vegas flows with a higher rate always reach the upper threshold (where a flow starts to decrease its rate) easier. However, the conservative addictive rate updates extend the convergence time. Copa assumes the flows observe accurate and the same queuing latency so that they converge to the same sending rate. However, the fluctuating nature of the latency (due to the burst traffic, scheduling strategy, *etc.*) determines Copa's difficulty in obtaining stable convergence results. Our experiments in the paper also show that the convergence of Copa is not stable.

Next, we discuss the algorithms with the thought of learning: PCC [4], Vivace [5], and BBR. PCC fails to guarantee fairness among flows due to its simple MIMD design. PCC employs the same fixed multiplier (default 1 ± 0.05) for all the competing flows to find the correct sending rate, which brings multiple convergence points. Vivace utilizes a gradient ascent method to amend the bad convergence performance of PCC. With this design, the competing flows always move toward fairness. But the convergence speed is not that satisfactory because the step size of Vivace is too conservative and has can not adapt to the available bandwidth. In BBR, if the bandwidth resource has been exhausted, the rate difference between two flows will be slightly reduced after an asynchronous bandwidth probing process.

D Distribution of the ACK Arrival Intervals

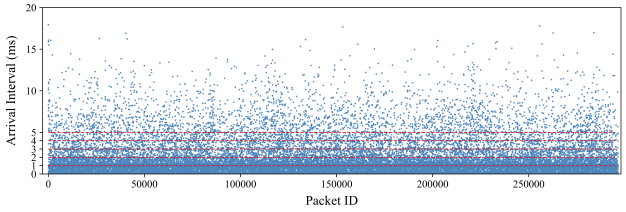


Figure 14: The distribution of ACK arrival intervals at a 100*Mbps* link. *x*-axis: the ID of the packet (about 30,0000 sampling points). *y*-axis: the time interval between each packet and its last packet. Most of the intervals are shorter than 2*ms*, which motivates us to utilize the ACK's arrival to trigger the data transmission.

Figure 14 shows the distribution of ACK arrival intervals at a 100*Mbps* link, with a total of about 30,000 sampling points. The *x*-axis is the ID of the packet, and the *y*-axis is the time interval between each packet and its previous packet. From this figure, we can observe that the statistical data is most dense within 1*ms*, *i.e.*, the arrival interval of most ACKs is less than 1*ms*. The points that are more than 5*ms* are sparse, and only a few of them exceed 10*ms*, which is the default highest precision of the operating system timers.

E Accuracy of RTT measurements

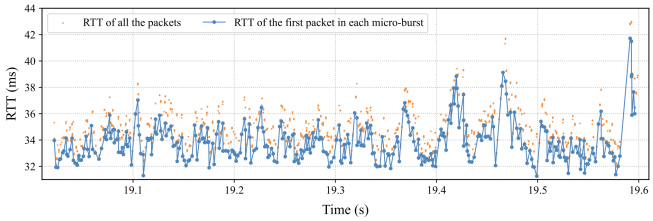


Figure 15: The RTT measurements of all the packets (the yellow points) and of the first packet in each micro-burst (the line points, which are connected through straight lines).

As Figure 15 shows, almost all the orange points (except those that coincide with the blue points) are above the blue line. This verifies our point that except for the first packet in each micro-burst, the subsequent ones are often affected by the extremely short RTT increase caused by the bursty traffic, resulting in a longer latency.

These higher latency measurements are usually regarded as the signal of congestion. However, the full bandwidth utilization only sustains for a short while and the average utilization can be low.

F What does micro-burst measure

ETC employs utilizes the receipt rate of N successive packets (micro-burst) to serve as an estimation of the bottleneck available bandwidth. Bottlenecks can exist not only at the routers in the network but also at the end hosts. In this section, we first try to use several simple network models to illustrate that the micro-burst is always shaped by the last bottleneck it passes through (no matter whether the bottleneck exists at the end-host or in-network routers) and thus inflects the available bandwidth of the real bottleneck. Then, our experimental results show that flows passing through the same bottleneck can obtain similar pulling rate signals.

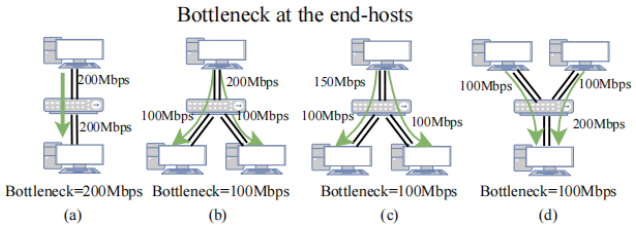


Figure 16: Network models where the bottleneck is at the end-hosts.

Figure 16 shows the network models where the bottlenecks are at the end-hosts. In Figure 16 (a), the bandwidth of the access link at the end-host and routers are the same (200Mbps). Therefore, the interval among packets keeps unchanged across the whole path. In Figure 16 (b) and (c), the access bandwidth of the link is higher than the access rate at the receiver, therefore, the receiver becomes the bottleneck. The measurement result should be 100Mbps (no cross traffic). In Figure 16 (d), the sender becomes the bottleneck. The micro-bursts can only be sent with a maximum rate of 100Mbps, which determines the instantaneous rate will not exceed 100Mbps.

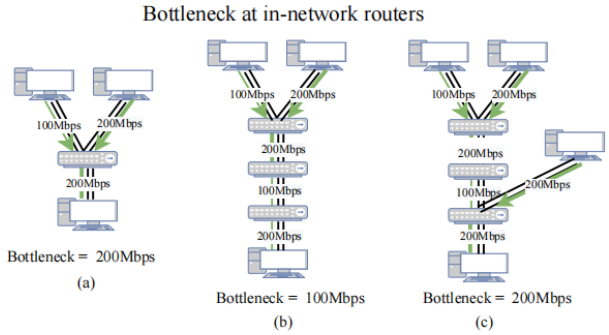
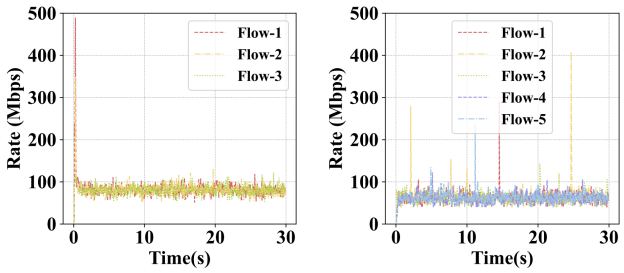


Figure 17: Network models where the bottleneck is at the in-network routers.

Figure 17 shows the network models where the bottleneck is at the routers. As Figure 17 (a) shows, the collective rate of the flows from two senders exceeds the rate of the egress port of the router, *i.e.*, the egress link of the router becomes the bottleneck. In Figure 17 (b), there are multiple bottlenecks. The intervals of the packets are shaped twice by the first and second routers, and the measurement result is determined by the second one. In Figure 17 (c), the micro-burst is shaped three times and determined by the third one even if the egress bandwidth of the third router is higher than the second one. To sum up, the micro-burst always measures the available bandwidth of the last bottleneck (as long as the micro-burst has been suspended in the queue), no matter whether the bottleneck is routers or end-hosts.



(a) R_p of three flows (b) R_p of five flows

Figure 18: The pulling rates measured by the senders that pass through the same bottleneck.

To validate that senders can obtain similar pulling rates, we conduct separate experiments with three and five flows originating from different sources but sharing the same destination. These flows were routed through a bottleneck router with a 200Mbps limit, while the source and receiver had no restrictions. Figure 18 presents the measured pulling rates collected at the sender. In Figure 18(a), the pulling rates measured from the three flows exhibit oscillations around 80Mbps, slightly exceeding the average available bandwidth of 66Mbps. Similarly, in Figure 18(b), the measured pulling rates from each flow fluctuate around 65Mbps, slightly surpassing the average available bandwidth of 40Mbps. Despite minor variations, the measured results from each flow remained consistent. However, a few outliers were detected, which were subsequently eliminated using a simple 3σ rule [43] for outlier removal.