Future Generation Computer Systems 149 (2023) 317–329

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Vivace-Distributed: A novel congestion control mechanism for JointCloud environments

Jianzhi Shi [a], Bo Yi [a,1], Xingwei Wang [a,*], Min Huang [b], Peichen Li [a], Chao Zeng [a], Keqin Li [c,2]

[a] College of Computer Science and Engineering, Northeastern University, Shenyang 110169, China
[b] College of Information Science and Engineering, Northeastern University, Shenyang 110819, China
[c] Department of Computer Science, State University of New York, NY 12561, USA

ARTICLE INFO

Article history:
Received 6 March 2023
Received in revised form 9 June 2023
Accepted 19 July 2023
Available online 22 July 2023

Keywords:
Congestion control
JointCloud
TCP friendliness
Distributed online learning
Cloud computing
Game theory

ABSTRACT

As the customer base of cloud computing services continues to grow, the demand for providing cloud resources in an economic manner has become increasingly important. To incorporate multiple clouds, JointCloud has been proposed for sharing cloud services. However, efficient communication among clouds is essential to enable resource sharing in JointCloud, which poses a critical challenge. The existing congestion control mechanisms for JointCloud have been founded to be inadequate in sharing network resources effectively and fairly, resulting in performance shortcomings. The maximum achievable transmission rate is limited to only 50% of the bandwidth, and the minimum and maximum transmission rates for different clouds in JointCloud may have a difference of up to ten times in magnitude.

We propose a novel distributed congestion control mechanism for the JointCloud environment, which leverages ideas from rich literature on *distributed online learning*. The proposed mechanism operates in a distributed manner, with an aggregative variable representing the network condition integrated into each sender's utility function. Our theoretical analyses establish that the proposed mechanism achieves a unique Nash equilibrium, enabling fair resource allocation among all senders. Furthermore, our simulation evaluations demonstrate that the proposed mechanism outperforms the existing congestion control mechanism in JointCloud environment by achieving a 20% higher throughput in the presence of at least 1% random loss, a 50% higher throughput for short TCP flows, and a throughput ratio of 1 when sharing bandwidth with conventional mechanisms, which indicates a perfect TCP friendliness towards other mechanisms. In summary, our proposed congestion control mechanism is the first to work in a distributed manner, which improves the efficiency and fairness of the congestion control mechanism. The proposed mechanism has the potential to significantly enhance the performance of communication among clouds, making a noteworthy contribution to cloud resource sharing.

1. Introduction

With economic globalization, more and more companies have joined the international trade of commodities and services, resulting in greater interdependence of the world economies [1]. A cloud computing service allows any terminal to access for the duration it requires, which makes it possible for the vast and distributed computing resources available on the network without worrying about the particular locations and internal structures of these resources. Although cloud computing services have emerged as a solution to provide users with always-on resources, the use of cloud computing services presents several challenges. One significant challenge is that, in many cases, many companies require bursts of resources that exceed the processing ability of a single cloud. For example, events like Taobao's "November 11", Walmart's "Black Friday" and many other wholesale events require more than ten times the resources of normal days, which puts great pressure on a single cloud. Another challenge arises from the emerging trend towards a shared economy. The globalized economy is undergoing a new evolution that advocates cooperation among multiple potentially competing entities rather than monopolization. As more entities participate in the globalized economy, it is unlikely that a single constraint can rule them

* Corresponding author.
E-mail addresses: 2210722@mail.neu.edu.cn (J. Shi), yibobooscar@gmail.com (B. Yi), wangxw@mail.neu.edu.cn (X. Wang), mhuang@mail.neu.edu.cn (M. Huang), lpc978677763@gmail.com (P. Li), zchneu@gmail.com (C. Zeng), lik@newpaltz.edu (K. Li).

[1] Member, IEEE.
[2] Fellow, IEEE.

https://doi.org/10.1016/j.future.2023.07.023
0167-739X/

all due to issues such as data ownership and political problems. This requires a new globalized cooperative computing model to support such globalized yet cooperative businesses.

Both researchers and companies are starting to take "new" cloud services into account to address solutions such as bursts demand of resources and constrained data processing problems caused by political problems. In response to this trend, JointCloud [2], a cross-cloud collaboration architecture that integrates different clouds' cloud services, is proposed. JointCloud draws on the concept of air alliance, aiming to enhance cooperation among multiple clouds and provide cross-cloud services. JointCloud takes an essential step towards providing cloud service in a cooperative manner, where all cloud providers collaborate to provide cloud services with low cost and high availability. Effective communication among clouds is crucial for JointCloud architecture. As communication between different clouds determines the resource scheduling rate, which directly impacts the quality of users' experience.

The increasing diversity of cloud services and more complex dependencies have made the traffic in JointCloud more dynamic and complicated, leading to greater pressure on the communication between clouds. To efficiently manage communication among clouds, a new congestion control mechanism that manages the sending rate according to the network condition is required. However, existing congestion control mechanisms manage network resources based on signals detected by themselves, disregarding cooperation among independent clouds, which often results in unfair resource allocation. In conventional congestion control mechanisms, packet loss and round-trip time (RTT) are considered decision signals. The entire decision-making process relies on these measurements and the pre-defined rule based on human understanding of the network. The conventional rule-based mechanisms are more susceptible to many unpredictable factors [3], resulting in poor performance, which brings down the efficiency in the use of JointCloud computing services, thereby, the serviceability of JointCloud.

Although the congestion control mechanism has been extensively studied for the last decades, no feasible solution has been proposed to handle congestion cooperatively based on other sender's action and care about its actions impact on the network condition. In other words, no feasible solution has been proposed to handle congestion cooperatively. However, in fields such as warehouse location problems, transportation systems, and signal processing, many researches tend to handle congestion in a cooperative manner. An agent's local decision rest not only on its own decision variable but also on some weighted average decision variables of others [4]. For example, in target surrounding, an agent's decision relies on the positions of other agents in addition to its own position. Motivated by these facts, We propose a new congestion control mechanism for the JointCloud architecture that manages network resources in a cooperative manner. Our proposed mechanism leverages ideas from *distributed online convex optimization* [5]. An aggregative variable building upon all senders' performance metrics is involved in each local utility function. To ensure data privacy, JointCloud Collaboration Environment (JCCE) computes the aggregative variable, which is unknown to any individual cloud. Our proposed mechanism enables JointCloud to handle congestion cooperatively, which can significantly improve the serviceability of the system. In summary, our contributions are as follows,

- We propose a congestion control mechanism for the JointCloud environment based on distributed online convex optimization, which enables cooperative avoidance or relief of congestion. Our proposed mechanism adopts a utility-based architecture where the throughput over a short time is computed by receivers and sent to JCCE for computing the aggregative variable, which is then involved in each sender's utility function. The aggregative variable represents the aggregated information of all decision variables. The utility is then translated into sending rate, in this manner, the proposed mechanism handles congestion cooperatively.
- We formulate sharing of network resources in JointCloud as a socially concave game and theoretically prove that there is always a unique Nash equilibrium for a proper choice of the utility function. We believe that Nash equilibrium is a suitable approach for allocating network resources because it ensures that each cloud maximizes its own utility while taking into account the actions of other clouds. In summary, Nash equilibrium provides a fair and effective way to allocate resources. Moreover, the Nash equilibrium guarantees the stability of the JointCloud environment and prevents any cloud from utilizing too much network resources, which could lead congestion and reduced performance.
- Extensive experiments with various congestion control mechanisms demonstrate that Vivace-Distributed outperforms existing congestion control mechanisms in terms of throughput and stability. Specially, Vivace-Distributed achieves 20% higher throughput when there is a random loss compared with the best existing congestion control mechanisms, while also a better stability. In the simulated JointCloud environment, Vivace-Distributed achieves a 50% higher throughput for short flows and a 40% faster convergence speed than conventional congestion control mechanisms. Additionally, Vivace-Distributed achieves a throughput ratio of 1 when sharing bandwidth with conventional congestion control, which indicates perfect TCP friendliness. The experimental results confirm the effectiveness of our proposed congestion control mechanism.

The rest of the paper is organized as follows. Section 2 provides a brief review of the related works. In Section 3, we give an overview of the proposed mechanism. Sections 4 and 5 present the utility function design and the aggregative variable respectively. In Section 6, we introduce how to convert utility values to sending rates. The simulations and performance evaluations are presented in Section 7. Finally, Section 8 concludes the paper, highlighting the contributions of our work and discussing future work.

2. Related work

2.1. Concept and architecture of JointCloud

JointCloud is a new type of cloud computing model based on the collaboration between service entities and deep integration of cloud service from multiple cloud. Serverless computing [6] are utilized to support collaboration between disparate clouds. Users or other clouds remotely invoke the different cloud resources and service capacities [7]. In this manner, the cloud can provide cloud services cooperatively, and developers are able to use cloud services without noticing which cloud provides the service.

JointCloud aims to build a cloud ecosystem; in this ecosystem, clouds provide cloud services independently while providing resources cooperatively when there is a burst of demand for resources. In order to support the collaboration between clouds, several mechanisms are proposed. The JointCloud architecture [2], shown in Fig. 1, provides a working process for JointCloud. JointCloud architecture has two essential parts: the JointCloud Collaboration Environment (JCCE) and the Peer Collaboration Mechanism (PCM). JCCE includes several BlockChain-based services [8–11] designed for transaction-related issues,

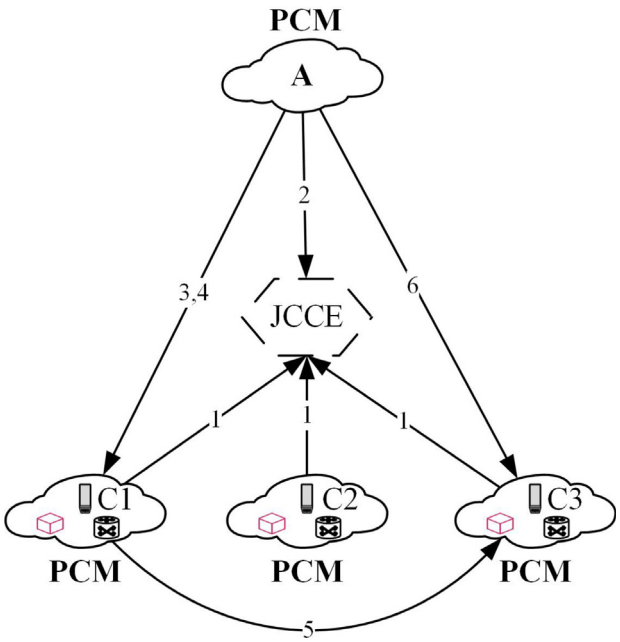


Fig. 1. JointCloud Architecture.

cloud cooperation [12], and evaluation of cloud services. Based on JCCE, as long as PCM is implemented between clouds. A PCM includes all protocols during cloud collaboration. Thus, if a cloud would like to join the JointCloud, it should implement the PCM to join the ecosystem.

For example, when a cloud tries to find the lowest-priced cloud resources in the JointCloud environment. Fig. 1 illustrates the process of renting cloud resources, 1. Clouds C1 to C3 transfer their price and types of resources at present to JCCE; 2. Cloud A sends a request for renting cloud resources and checks the price on JCCE; 3. Cloud A deploys the task to cloud C1; 4. Due to price fluctuations, cloud A sends a request to cloud C1 to transfer the task to cloud C3; 5. C1 delivers the task to C3; 6. Cloud A sends a request to C3 to start the task.

2.2. Communication among clouds and development of congestion control

For the communication between clouds, tunneling techniques are adapted by some hybrid solutions (like [13,14]) to connect the private cloud to the public cloud. For QoS, the current carrier network reverses most of the existing bandwidth to handle bursts of traffic [15]. For the inter-datacenter network, the current carrier network always measures the inter-datacenter link at the edge, and network-involved tasks are always scheduled carefully according to the network condition to utilize the bandwidth better. For other methods, Yuan et al. [16] proposes a delay-tolerant data congestion control mechanism for cloud system. COSCO2 [17] predicts the upcoming workload and reduces extravagant power consumption at cloud data centers. However, if different clouds are willing to use these solutions, they need to implement the same management software, which limits the deployment of these methods in a large scale. Thus, more clouds still utilize the standard TCP protocol suite for communication currently.

The performance of the TCP protocol depends heavily on the interactions between the underlying network and the transport layer. Congestion Control, as a critical mechanism, limits the number of packets injected into the network. A practical congestion control mechanism should achieve numerous goals [18]. Firstly, a congestion control mechanism should be able to efficiently manage network resources to prevent congestion collapse, which is the main reason for the existence of CC. Secondly, a good CC mechanism should ensure high bandwidth utilization and a fair share of available network resources among all competing flows sharing the same bottleneck. Thirdly, a CC mechanism should guarantee quick convergence to a stable point.

Conventional rule-based congestion control mechanisms (e.g., [19–23]) often fail to converge to a stale point. These mechanisms use some signals to detect congestion and then adjust the CWND or sending rate according to some pre-defined rules. For example, Cubic, Reno and New Reno are loss-based mechanisms that detect congestion through packet loss and adjust the CWND using additive increase multiplicative decrease (AIMD) algorithm. BBR estimates RTT and bandwidth, aiming for a CWND equal to the bandwidth-delay product. However, these mechanisms may fail to adapt to different networks and result in poor performance. Recently proposed CC mechanisms (e.g., [24–28]), including Remy [24], PCC [25], PCC-Vivace [26] and Learning-TCP [27] apply Machine Learning (ML) techniques to congestion control. Learning-TCP utilize learning automata to improve the performance of TCP over ad hoc wireless networks. Remy replaces the human designer with a map between RTT and sending rates, which is trained offline. However, The need for offline training makes it unable to adapt to changing network conditions [29]. In order to solve the problem existing in Remy, PCC utilizes online learning. A PCC sender sends at a certain rate and observes performance metrics; sending rate is then adjusted in a direction empirically associated with higher throughput. However, PCC still has not exploited the full potential of online learning. Thus, PCC-Vivace applies ideas from the gradient-descent algorithm to congestion control. Although online learning can respond to changing network conditions quickly, the performance of the agent may be unstable as their greedy exploration may get stuck at a local optimum in some cases [30]. It should also be noted that online learning usually has a long convergence time [31], which fails to satisfy the need for short flows. By considering the architecture of JointCloud, we propose a congestion control mechanism that works in a distributed manner, achieves higher throughput and stability, and meets the demand for a cooperative congestion control mechanism.

3. Overview of Vivace-Distributed

In this section, we will present the overall design of Vivace-Distributed and explain why Nash Equilibrium is a suitable approach for sharing network resources. As shown in Fig. 2, Vivace-Distributed consists of four modules, a sending module, a monitoring module, a utility module, and a rate control module. The sending module is deployed on the Control Panel of the PCM and is responsible for handling the details of communication between clouds by controlling the cloud's sending rates. The remaining three modules are deployed on Information Panel; which facilitates communication among different panels in PCM and communicate with JCCE. During data transmission, the monitoring module collects packet-level events and aggregative variables from JCCE; The utility module then summarizes these metrics in the form of a numeric utility value, which is associated with the corresponding sending rate in the rate control module. Based on the relationships between different sending rates and their corresponding utility values, the rate control module adjusts sending rates towards the direction that maximizes utility. To test different sending rates, Vivace-Distributed divides time into successive time periods called *Monitor Intervals* (MIs), which are typically one or two RTTs in length.

As shown in Fig. 2., rate control module is activated when a sender starts sending data. The sender sends data at an initial

rate, and as packets are received by the receiver, SACK is sent back to the sender. The monitoring module utilizes the timestamp to compute RTT, loss rate and other performance metrics in this interval. Simultaneously, the receiver computes the throughput in this interval and sends it to JCCE. JCCE computes the aggregative variable from the performance metrics of receivers, the aggregative variable is then sent to each sender. After the monitoring module has received all the performance metrics, the utility module computes the utility value in this interval, and sends it to the rate control module for deciding the next interval's sending rates. Each sending module adjusts the sending rate in the direction that maximize its own utility value.

We adopt the utility-based approach attributed to its high availability. The utility module allows for utility functions based on different performance metrics, which can be determined based on the requirements of different cloud services. The flexibility in selecting performance metrics ensures high availability with minimal overhead.

To share network resources in a cooperative manner, Vivace-Distributed introduces aggregative variable [5]. We formulate sharing of network resources among clouds as a social game, in which all clouds try to maximize their own utility values. The proposed algorithm utilizes global online optimization to reach the Nash Equilibrium of the social game, where local information is exchanged among clouds. However, it is important to note that each individual cloud can only access partial information about the global problem, which may be private to the cloud. Therefore, it is important to choose a mechanism that supports global cooperation while guaranteeing data privacy. Thus, we choose JCCE to compute the aggregative variable since it is designed to support global cooperation, while guaranteeing data privacy. The clouds send their own decision variables to JCCE, the aggregative variable is then computed by the JCCE and sent back to each cloud which is later involved in each local utility value.

Nash Equilibrium is a concept in game theory that describes a state in which each player makes a decision based on their own interests, taking into account the decisions of other players. Nash equilibrium is suitable to network resources sharing, because under the Nash equilibrium, each sender's strategy is the best response to the strategies chosen by the other senders. In other words, this means that no sender can improve their utility value by unilaterally changing their strategy, given the strategies of the other players. Nash Equilibrium is a suitable for sharing resources for two main reasons. Firstly, the mechanism works cooperatively, with the aggregative variable representing the decisions of other players, which is consistent with the scenario described by the Nash Equilibrium. This ensures that the allocation of network resources can always achieve a stable point. Secondly, Nash Equilibrium provides a fair and efficient way to allocate resources. Each cloud can make decisions based on its own utility function, which can be determined by its specific needs and constraints. Moreover, the unique Nash equilibrium prevents any cloud from deviating from the equilibrium strategy to gain an advantage, which could lead to congestion and reduce performance. Therefore, using Nash Equilibrium to model network resources sharing is a good approach that ensures fairness, efficiency and stability.

Under the architecture described above, two problems arise, which are discussed in the following sections. Firstly, we need to design a new utility function with an aggregative variable. For our new utility function, we employ a game-theoretic analysis and show that a unique Nash Equilibrium always exists, which meets the requirements for a fair share of network resources. Secondly, we need to choose an appropriate variable to represent the global state of the network. We choose the average of all clouds' throughput as the aggregative variable reflects the overall performance of the network while also be simple to compute. We also evaluate the performance of the aggregative variable theoretically.

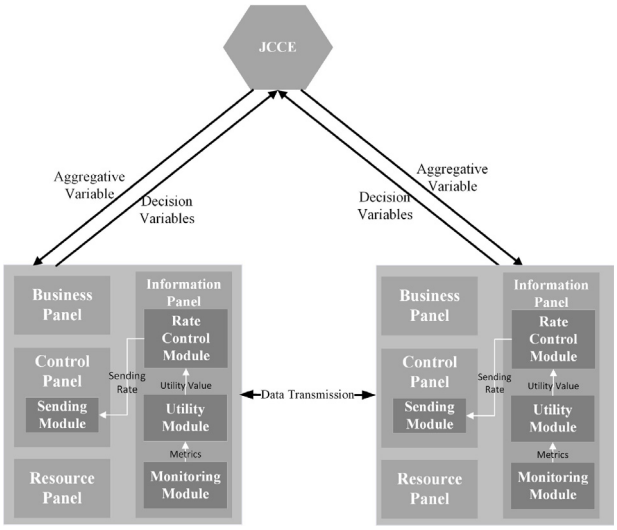


Fig. 2. Architecture of Vivace-Distributed.

4. Utility design of Vivace-Distributed

In this section, we will introduce the utility function utilized by Vivace-Distributed. We first introduce the key metrics employed by the utility function, then analyze the existence of the Nash Equilibrium and the stability of the utility function upon convergence.

4.1. Utility framework of Vivace-Distributed

Vivace-Distributed divides time into successive Monitor Intervals (MIs). Clouds use the utility function in Eq. (1) to convert the performance metrics collected at the MI into a numerical utility value,

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

In the utility function, t, d, b, c are constants, x_i represents the sending rate of cloud i, T_{avg} is the average throughput of all clouds in this MI, and L_i is loss rate observed in this MI. The term $d(RTT_i)/dT$ represents the "RTT gradient" which indicates how latency changes. Average throughput is integrated into the utility function as a new reward since the utility function is too sensitive to latency and packet loss. We intend to add a new reward to the utility function to reduce the impact of the RTT gradient.

To model the competition among clouds, we consider a network with a set of N of n clouds. Let $X = (x_1, \ldots, x_n)$ denote the vector that describes all clouds' sending rates, C represents the capacity of the bottleneck link, and S represents the total sending rate of all the senders ($S = \sum_{i\in N} x_i$). To simplify the proof, we assume that the packet size is 1. Therefore, the average throughput can be defined by,

$$T_{avg} = \begin{cases} S/n, & \sum_{i\in N} x_i < C \\ C, & otherwise \end{cases} \tag{2}$$

Assuming the tail drop queue, the bottleneck queue size changes at a rate of $S - C$, since the queue drains at a rate of C, the term $d(RTT_i)/dT$ can be defined by,

$$\frac{d(RTT_i)}{dT} = \frac{\sum j_{\in N} x_j}{C} \tag{3}$$

The packet loss can be described as follows,

$$L_i = \begin{cases} \left(1 - \frac{C}{\sum_{i \in N} x_i}\right), & \sum_i x_i > C \\ 0, & otherwise \end{cases} \tag{4}$$

Hence, the utility function can be described as follows,

$$u = \begin{cases} x_i^t + d\frac{S}{n} - bx_i \frac{\sum_{j \in N} x_j - C}{C}, & \sum_i x_i < C \\ x_i^t + dC - bx_i \frac{\sum_{j \in N} x_j - C}{C} - cx_i\left(1 - \frac{C}{S}\right), & otherwise \end{cases} \tag{5}$$

4.2. Nash equilibrium

Definition 1. A socially concave game is defined by the following properties of the player's utility function:

1. Each player's utility is convex in the player's strategy.
2. The sum of the utilities of all players is concave in the strategies of all players.
3. The utility of each player is convex in the strategies of the other players.

A strictly socially concave game [32] is a subclass of socially concave games in which the utility function satisfies at least one condition of a socially concave game in a strict manner, which means that either strict convexity or strict concavity is satisfied. A strictly socially concave game admits a unique Nash equilibrium, ensuring that the game converges to a stable point.

Definition 2. A no-regret algorithm is an algorithm that guarantees an average payoff that is at least the payoff induced by the optimal action in hindsight.

Zinkevich [33] proved that a gradient descent algorithm with a bounded derivative defined on a convex utility function satisfies the no-regret property. The no-regret property has essential implications for the design of algorithms, as it provides a theoretical guarantee of convergence to the Nash Equilibrium.

Theorem 1. *When there are n Vivace-Distributed senders sharing a bottleneck link in the network if each sender's utility function is defined as Eq. (1), all senders' sending rates converge to the configuration* $(x_1^*, x_2^*, \ldots, x_n^*)$ *which satisfies* $x_1^* = x_2^* = \cdots .x_n^*$.

The utility function is composed of performance metrics and parameters. As for the choice of performance metrics, while other performance metrics can be utilized to the utility function as long as they are relevant to the property being evaluated. However, we believe that RTT-gradient, loss rate and throughput are suitable for congestion control, as these metrics provide an intuitive representation of the network condition. As for the choice of values for parameters, d, b, c and t in the utility function have crucial implications for the existence of the Nash Equilibrium when multiple senders compete. In summary, the properties stated in Theorem 1 are based solely on the properties of the player's utility function, only the choice of utility functions is determined exogenously. Although other utility functions can also be utilized, we believe that the proposed utility function is appropriate for the JointCloud environment.

Proof. A Strictly socially concave game admits a unique Nash Equilibrium. Moreover, Even-Dar et al. [34] proved that any no-regret [35] algorithm in a socially concave game will converge to the Nash equilibrium of the game. Therefore, it is essential to demonstrate that the three properties outlined in the theorem are satisfied.

1. The first property that the utility function of each player is convex in the player's own strategy can be proved by taking the second derivative of the utility function. The second derivative of the utility function is,

$$\frac{d^2u(x_i)}{d^2(x_i)} = t(t-1)x_i^{t-2} - 2\frac{b}{C} - cC\frac{2S(S-x_i)}{S^4} \tag{6}$$

It is obvious that When $0 < t < 1$, the utility function is convex in its strategy. Therefore, the constraint on t is $0 < t < 1$.

2. The sum of all utility functions is strictly concave in $X = (x_1, x_2, \ldots, x_n)$. The sum of the utility function is,

$$\begin{aligned} U(X) &= \sum_i u(x_i) = \sum_i (x_i^t + dT_{avg} - cx_i(1 - \frac{C}{S}) \\ &\quad - bx_i\frac{\sum_{j \in N} x_j - C}{C}) \\ &= \sum_i x_i^t + dnS - cS\left(\frac{S-C}{C}\right) - bS\left(1 - \frac{C}{S}\right) \end{aligned} \tag{7}$$

Since S is a linear function of X, we only need to prove that $U(X)$ is concave in S. The second derivative of $U(X)$ is,

$$\frac{d^2U(X)}{dS^2} = \frac{d^2\sum_i x_i^t}{dS^2} - \frac{2(b+c)}{C} \tag{8}$$

Based on the proof of the first property, we know that $0 < t < 1$. In this instance, x_i^t is a concave function. Thus, we can conclude that $d^2\sum_i x_i^t/dS^2 < 0$. The second term $-2(b+c)/C$ is a negative constant. Therefore, the utility function is strictly concave in $X = (x_1, \ldots .x_n)$.

3. In other senders' strategies, the utility function is convex. Let x_{-i} denotes $x_{-i} = (x_1, \ldots .x_{i-1}, x_{i+1}, \ldots, x_n)$, and r_i represents $r_i = \sum_{j \neq i} x_j$. Since r_i is a linear variation of x_{-i}, thus, the utility function only needs to be convex in r_i. To simplify the subsequent proof, we rewrite the utility function as follows,

$$u(x_i, r_i) = x_i^t + d\frac{C}{n} - bx_i\frac{x_i + r - C}{C} - cx_i\left(1 - \frac{C}{x_i + r_i}\right) \tag{9}$$

We take the second derivative,

$$\frac{d^2u(x_i, r_i)}{dr_i^2} = cx_i\frac{C}{S^3} \tag{10}$$

Since $cx_iC/S^3 > 0$, it is obvious that property 3 is held in a strict manner.

Therefore, our proposed utility function satisfies the conditions of a strictly social concave game, resulting in a unique Nash Equilibrium. This ensures that the mechanism is effective and efficient when multiple sender competing for network resources.

4.3. Stability upon convergence

Ideally, during convergence, the latency should not exceed the base RTT when the buffer is empty. Theorem 2 demonstrates how Vivace-Distributed provides a sufficient condition for avoiding latency inflation at convergence. This condition ensures that the congestion control mechanism achieves a low latency, leading to a high performance.

Theorem 2. *The latency measured at the stable point is the base RTT of the link if the term b and d satisfies* $(b - d) \geq tn^{2-t}C^{t-1}$

Proof. First, it is evident that if b is a large constant, the latency would be a large penalty in the utility function, and any increase

in latency would lead to a large decrease in the utility value. In particular, an ideal coefficient b can make the sum of all clouds' rate will always decrease. As the change in the sending rates of a cloud is proportional to the change in the gradient of its utility function, the sum of all clouds' sending rates changes is in proportion to the sum of all the gradients of the utility of all clouds. Hence, we use x_i^S to represent cloud i's sending rate at the beginning of MI, and x_i^E denotes the sending rate of cloud i at the end of the MI. We aim to find a value of b such that $\sum_i (x_i^E - x_i^S) < 0$. We make the assumption that the buffer is not empty upon convergence since when the buffer is empty, there is obviously no latency inflation (there is even no latency). Therefore, we take the sum of all utility gradients, and set it to be less than zero to ensure a reduction in the sum of sending rates.

$$\begin{aligned}
&\sum_i (x_i^E - x_i^S) \propto \sum_i \frac{du_i}{dx} \\
&= \sum_i \left(tx_i^t - i + \frac{d}{n} - \frac{d}{dx_i} \left(\frac{bx_i(x_i + r_i - C)}{C} \right) \right) \\
&= \sum_i \left(tx_i^{t-i} + \frac{d}{n} - b\frac{(x_i + r_i - C)}{C} - b\frac{x_i}{C} \right) \\
&\leq \sum_i \left(t\left(\frac{S}{n}\right)^{t-1} + \frac{d}{n} - b\frac{S}{C} + b - b\frac{x_i}{C} \right) \\
&= tn^{2-t}S^{t-1} - b\frac{(n+1)S}{C} + nb + d
\end{aligned} \tag{11}$$

Therefore,

$$tn^{2-t}S^{t-1} - b\frac{(n+1)S}{C} + nb + d < 0 \tag{12}$$

And then,

$$\frac{tn^{2-t}S^{t-1} + d}{\frac{S(n+1)}{C} - n} < b \tag{13}$$

In order to find the lower bound of parameter b, we set $S = C$ since the queue is always not empty, so $S > c$.

$$tn^{2-t}C^{t-1} + d \leq b \tag{14}$$

$$tn^{2-t}C^{t-1} \leq (b - d) \tag{15}$$

5. Analyses of the aggregative variable

5.1. The selection of aggregative variable

As discussed in Section 1, an ideal performance metric for the aggregative variable should not only be an indication of a single cloud's congestion for the time being, but also consider competition among clouds. The implication is twofold. First, when a TCP connection between clouds is freed, the aggregative variable should encourage existing connections to take over under-utilized capacity. Second, when a new TCP connection is established, the bottleneck buffer is already full; and the selected aggregative variable should identify and react to the newly established TCP connection, freeing up capacity taken by the existing connection. Finally, the aggregative variable should ensure quick convergence to the stable sending rate. Based on that intuition, we have chosen average throughput as the aggregative variable. Average throughput is calculated as the average of all clouds' throughput within an MI, and it works as an intuitive indicator for flow competition. As if one flow experiences unfairness or high packet loss, the average throughput decreases sharply.

$$\tilde{Throughput} = \sum_i (Delivered_i)/n \tag{16}$$

n represents the number of clouds sharing the capacity, and $Delivered_i$ represents the amount of data transmitted successfully (i.e., data that has been acknowledged by the receiver) by cloud i.

Achieving maximum bandwidth in the shortest possible round trip time is crucial due to the availability of gigabit bandwidth. Recent research [36] has revealed that most short flows, such as web search, fail to reach their highest sending rates, resulting in under-utilization of available bandwidth. We have observed that the utility-based architecture slows down the convergence speed. Assuming a cloud attempts to send data at a rate that the queue is built, but far from full, then there exists a queuing delay; which will be penalized due to the increase RTT. As a result, the cloud slows down the rate's increasing speed and results in a slow convergence. Thus, we aim to find the aggregative variable that acts as a reward in the utility function during the initial stage of a TCP connection, while simultaneously preventing a flow from taking too much capacity when new TCP connection is established.

Average throughput is a suitable performance metric as it captures good throughput and reflects the usage of network resources. As long as the flows actively increase the sending rates instead of blindly speeding up the transmission, the average throughput increases leading to a higher utility value. For example, when competing flows stop transmitting, bandwidth is freed up, resulting in a decrease in the RTT gradient and an increase in utility value, which drives the flows to take under-utilized capacity. As a result, the average throughput increases as network resources allocated to a flow increases. This leads to a faster increase in utility value, resulting in a faster convergence to stable point. In addition, average throughput is also suitable in scenarios where a new TCP connection is established between clouds. For instance, when several flows already reached a stable point, the buffer is already full, and most network resources are already allocated to existing flows, a new TCP connection is established. The newly established TCP connection will induce an increase in the number of existing TCP connections, resulting in a significant decrease in average throughput. In such cases, a sufficiently large penalty will lead the clouds to reduce their sending rates and free up capacity for the newly established connection. Overall, average throughput serves as a reliable indicator for network resources, while, it is important to note that average throughput may not be ideal for all requirements of an application. But we believe that average throughput is a suitable aggregative variable for most scenarios.

5.2. Performance analysis of the aggregative variable

Vivace-Distributed' rate control begins with a startup phase in which the sender doubles its sending rates. When the utility value gradient becomes negative for the first time, Vivace-Distributed exists slow start phase enters the online learning phase. Once the slow start phase completes, Vivace-Distributed will never enter the slow start again. Comparing the two phases, sending rates increase much faster in the slow start phase. Thus, we can conclude that a longer slow start phase results in a shorter time to reach the stable point. Therefore, we try to extend the duration of the slow start phase through the aggregative variable. We theoretically prove that, compared with the previous online-learning based congestion control mechanism, Vivace-Distributed has a longer slow start phase.

According to convex optimization theory, the first time the utility function decreases, the first derivative of the utility function is equal to 0. We take take the first derivative of the utility function,

$$\frac{du(x_i)}{dx_i} = tx_i - b\frac{d(RTT_i)}{dT} - cL_i \tag{17}$$

Algorithm 1 Vivace-Distributed.

Input: sending rate of sender i in interval M: x_i^M
Output: sending rate of sender i in interval $M+2$: x_i^{M+2}

Randomly initialized
start sending packets
update RTT_i^M,$RTT\_gradient_i^M$ L_i^M, $Throughput_i^M$
if Sending rate decrease throughput increase **then**
  Try the same sending rate again
end if
send $Throughput_i^M$ to the JCCE
if aggregative_variable == 0 **then**
  Keep waiting aggregative variable sent from the JCCE
end if
if $RTT\_gradient_i^M < flt_{latency}$ **then**
  $RTT\_gradient_i^M = 0$
end if
calculate the utility value u_i^M
set sending rate in the next interval $M+1$: $x_i^{M+1}(1+\epsilon)$
calculate utility value u_i^{M+1}
set sending rate in the next interval $M+2$: $x_i^{M+2}(1-\epsilon)$
calculate utility value u_i^{M+2}
update the utility gradient: $\gamma_i^M = (u_i^{M+2} - u_i^{M+1}/2\epsilon x_i^M)$
if $(\gamma_i^{M-1} > 0 \wedge \gamma_i^M > 0)$ **then**
  $\tau = \tau + 1$
  $\Delta r = (k * \tau + b) * \gamma_i^M$
  $Threshold = (\omega_0 + k * \delta) * r$
  if $\Delta r > Threshold$ **then**
    $\delta = \delta + 1$
    $x_i^{M+3} = x_i + Threshold$
  else
    $x_i^{M+3} = x_i + \Delta r$
  end if
else if $\gamma_i^M * \gamma_i^{M-1} < 0$ **then**
  $\tau = 0$
  $\Delta r = (k * \tau + b) * \gamma_i^M$
  $Threshold = (\omega_0 + k * \delta) * r$
  if $\Delta r > Threshold$ **then**
    $x_i^M + 2 = x_i + Threshold$
  else
    $x_i^M + 2 = x_i + \delta * r$
  end if
  $\delta = 0$
else if $\gamma_i^M < 0 \wedge \gamma_i^M - 1 < 0$ **then**
  $\tau = \tau + 1$
  $\Delta r = (k * \tau + b) * \gamma_i^M$
  $Threshold = (\omega_0 + k * \delta) * r$
  if $\delta r > Threshold$ **then**
    $\delta = \delta + 1$
    $x_i^M + 2 = x_i + Threshold$
  else
    $x_i^M + 2 = x_i + \delta * r$
  end if
end if

Then, in theory we can determine the sending rates when Vivace-Distributed exists slow start phase.

$$tx_i^{t-1} + \frac{d}{n} - b\frac{d(RTT_i)}{dT} - cL_i = 0$$
$$x_i = \sqrt[t-1]{\frac{b\frac{d(RTT_i)}{dT} + cL_i - d/n}{t}} \tag{18}$$

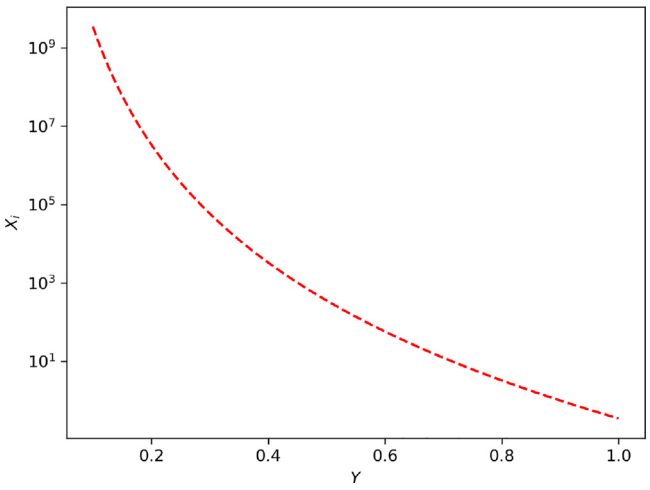


Fig. 3. Function Image of Eq. (19) (t = 0.9).

The term d/n is determined by parameter d and the number of flows.

Theorem 3. *The duration of the slow start phase is determined by the parameter d, where a larger value of d results in a longer slow start phase.*

Proof. To simplify the analysis, we rewrite Eq. (18). as follows,

$$x_i = \sqrt[t-1]{\frac{Y}{t}} \tag{19}$$

In this case, Y is regarded as the independent variable of the function, and according to Theorem 1., the value of t lies between 0 and 1. The image of the new function is depicted in Fig. 3 ($t = 0.9$, which is used for our simulation experiments).

As depicted in Fig. 3, the function value experiences a rapid decline within the range of 0–1, and becomes negative when $Y > 1$. Therefore, it is apparent that,

$$0 < b\frac{d(RTT_i)}{dT} + cL_i - \frac{d}{n} < 1 \tag{20}$$

Therefore, the parameter d/n meets the following condition,

$$0 < \frac{d}{n} < 1 \tag{21}$$

It can be observed from Fig. 3. that a smaller value of $b * d(RTT_i)/dT + cL_i - d/n$ corresponds to a higher threshold. A larger value of d results in a smaller value of $-d/n$, which in turn leads to a higher threshold. This implies that the user will exist the slow start phase with a higher sending rate, resulting in a shorter time required to reach the saturate bandwidth, leading to high utilization. Therefore, the parameter d plays a crucial role in determining the duration of the slow start phase. When the number of senders n is small, the impact of d is significant. However, as $n \to \infty$, d/n approaches 0, which reduces the impact of d and prevents the sender from naively increasing sending rates.

6. Transition from utility values to sending rates

To convert the utility value into the sending rate, Vivace-Distributed first computes the gradient of the utility function. Assuming Vivace-Distributed's sending rate in the interval is r, then Vivace-Distributed will adjust the rate of the following two

intervals to $r(1 \pm \epsilon)$ where ϵ is the step size which is always a conservatively small number. In the subsequent two MIs, Vivace-Distributed computes the corresponding utility value u_1 and u_2, then the utility gradient is then calculated as,

$$\gamma = \frac{u_1 - u_2}{2\epsilon r} \tag{22}$$

Vivace-Distributed then employs γ to determine the direction and magnitude of the rate change. To handle measurement noise, Vivace-Distributed employs a "confidence amplifier" to convert γ into rate changes.

The confidence amplifier starts with a relatively low "conversation factor", which increases as the amplifier gains "confidence" from its decision. The confidence amplifier works as follows,

$$\begin{aligned} r_{new} &= r + \theta\gamma \\ \Delta r &= \theta\gamma \\ \theta &= m(\tau)\theta_0 \\ m(\tau) &= k\tau + b \end{aligned} \tag{23}$$

Specifically, θ_0 is a conservatively small value, $m(\tau)$ is a linear function of τ with constants k and b. If a sender's rate changes in the same direction for τ consecutive MIs, θ is updated to $m(\tau)\theta$; when the direction of the sending rate change has changed, τ is reset to 0.

Additionally, the initial step size of sending rate may be potentially large, which may lead to an explosive growth in sending rate(i.e., from 1 Mbps to 80 Mbps), causing long latency and high loss rates. To prevent such overshooting of the bottleneck link's capacity, A dynamic change boundary is introduced to limit the growth in sending rate.

The change of sending rates is bounded by a threshold ωr; when rate change exceeds ωr, the upper limit of the actual rate change is ωr. The threshold is computed as follows,

$$\begin{aligned} Threshold &= \omega r \\ \omega &= \omega_0 + k * \delta \end{aligned} \tag{24}$$

The parameter ω is initialized to ω_0. After the threshold makes δ consecutive changes in the same direction, the ω is updated to $\omega = \omega_0 + k*\delta$, where k is a constant. The threshold increases when the rate change exceeds the threshold and decreases when the rate change is smaller than the threshold. When the direction of rate adjustment changes, the k is reset to 0. Above all, the specific execution process of Vivace-Distributed is shown in Algorithm 1.

Accurate RTT and loss rate measurements typically require a long observation time; however, this will slow the reaction to the changing environments. To address challenge, We have implemented a few mechanisms.

First, we set a threshold for RTT_gradient, as non-congestion-induced latency jitters always occur. The utility function used in Vivace-Distributed is sensitive to latency, and latency jitters may lead to a misinformed decision. Therefore, we have set a threshold for latency, where RTT gradient smaller than the threshold $flt_{latency}$ will be treated as 0. This loss-pass filtering mechanism helps to ignore small latency jitters caused by noise.

Second, we design a double-checking mechanism to address exceptional cases. In General, we avoid predicting network changes, but even under a dynamic network, a higher sending rate is unlikely to be the reason for a lower packet loss; more likely, such an observation is caused by measurement noise. To address these exceptional cases, Vivace-Distributed will re-run the same sending rate in the next interval; if the same condition occurs again, Vivace-Distributed will utilize the abnormal measurement; otherwise, it will abandon the measurement and utilize the measurement from the previous interval. This double-checking mechanism helps to ensure that the observed performance metrics are not due to measurement noise and the adjustment are accurate.

Table 1
Vivace's rate control default parameters.

Parameter	Value
MI duration	1 RTT
sampling step ϵ	0.05
initial conversion factor θ_0	1
initial dynamic boundary ω_0	0.05
dynamic boundary increment δ	0.1
confidence amplifier m(τ)	$\tau(\tau \leq 3)$, $2\tau - 3(\tau > 3)$

7. Evaluation

We implement the design of the previous sections and set the parameter t, d, b, c based on theoretical analysis. We set $t = 0.9$ to ensure that $t < 1$. We set b=1900 to achieve no inflation latency when a thousand senders share a bottleneck. We set $c = 11.35$ to avoid the impact of up to 5% random loss, as proposed in PCC-Vivace. We set $d = 1000$ to achieve the theoretically fastest convergence speed when there are 1000 senders. We believe that these parameter values are common set in real-world internet scenarios, and note that the parameters can be adjusted to meet the requirements of other networks. Unless stated otherwise, our other parameters use the values in Table 1.

Setup. We conduct experiments in emulated JointCloud environments, where one machine acts as JCCE, and other machines act as clouds. We use different flows to emulate different services, i.e., short flows are used for web searches, and long flows are used for the live migration of virtual machines. We implement CUBIC, BBR, and Reno in Linux kernel v4.10 [37]. We then evaluate the performance of these algorithms in terms of throughput, convergence time and other metrics under various network conditions.

7.1. Utility function

We set up a bottleneck link with 100 Mbps bandwidth, 30 ms RTT, 75 KB buffer, and varying random loss rates using Emulab [38]. The utility value of Vivace-Distributed under different random loss rates is shown in Fig. 4. As described above, during the slow start phase, Vivace-Distributed doubles the sending rate every MI, resulting in a rapid increase in the utility value. However, the utility value under a higher loss rate is smaller than that of a lower loss rate due to non-congestion loss. The random loss is a penalty to the utility value and reduces the increasing speed; however, the buffer is far from full, and the utility value keeps increasing, leading to increasing sending rates. However, at around the 27th interval, the buffer becomes full, and nearly all packets are dropped in this interval resulting in a negative utility value. As a result, the utility value decreases for the first time, and Vivace-Distributed exists the slow start phase and enters the online learning phase. As shown in Fig. 4, Vivace takes around 42 intervals to reach the stable point under 3% loss and around 66 intervals under 5% loss. At around 3 s, Vivace-Distributed converges to the stable point, and the utility value under 5% loss is smaller than that of the lower loss due to the higher non-congestion loss. As expected, the RTT gradient approaches 0 at the stable point; Vivace-Distributed tries to reduce its rates to reduce loss and thus achieves a lower sending rate and, as a result, a lower utility value.

Occasionally, measurement noise can cause abnormal increases in utility value; as illustrated above, the confident amplifier prevents Vivace-Distributed from varying a lot in sending rates (e.g., from 70 Mbps to 150 Mbps). The confident amplifier helps to stabilize the utility value by reducing the impact of measurement noise.

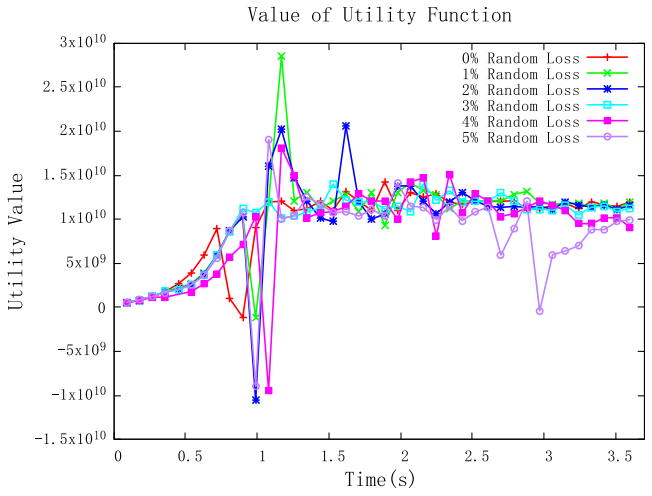


Fig. 4. Value of utility function.

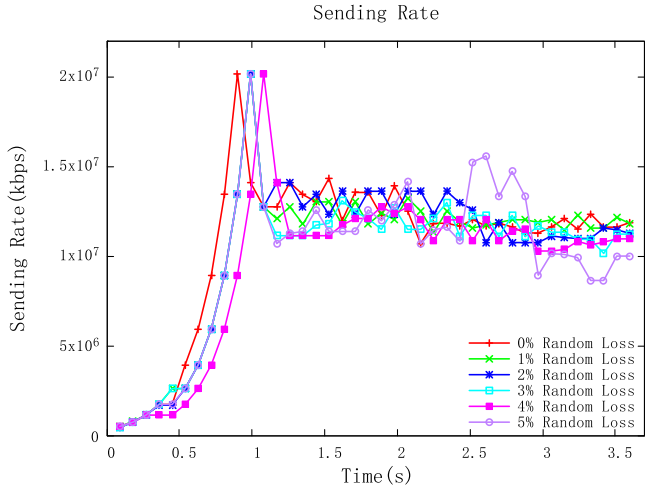


Fig. 5. Sending rate.

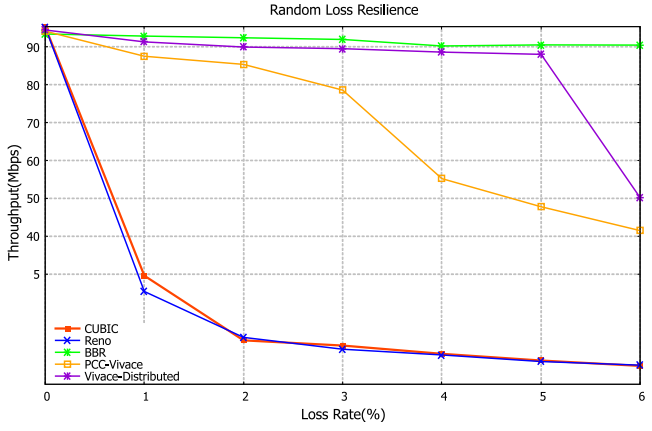


Fig. 6. Random loss resilience.

7.2. Sending rates

We use Emulab to set up an emulated link with 100 Mbps bandwidth, 30 ms RTT, 75 KB buffer, and varying random loss rates. Fig. 5 shows the sending rates achieved under different random loss rates. As shown in Fig. 5, Vivace-Distributed doubles its sending rates in the slow start phase. Corresponding to the utility value of Vivace-Distributed shown in Fig. 4., at about 0.9 s, Vivace-Distributed exists slow stars, and the sending rates drop to 130 Mbps, which also exceeds the bottleneck link's capacity. After that point, the buffer is full, and packet loss occurs, resulting in a sharp decrease in sending rates to drain the queue. Then Vivace-Distributed enters the online learning phase; during which it takes about 27 intervals to reach the stable point under 3% loss. At the stable point, Vivace-Distributed sends at around 110 Mbps, which is able to utilize all the bandwidth and the buffer. Under 5% loss, Vivace-Distributed takes about 40 intervals to reach the stable sending rate. Due to non-congestion loss, Vivace-Distributed's sending rates jitters, resulting in a long time to the stable point.

As we analyzed in Section 5, the aggregative variable extends the duration of the slow phase; Vivace-Distributed exists the slow start phase in a sending rate that exceeds the capacity of bandwidth and utilizes all network resources. Although Vivace-Distributed works aggressively, the aggregative variable acts as a significant penalty when there are other flows in the network; Vivace-Distributed can also achieve perfect TCP friendliness.

Our experimental results demonstrates that Vivace-Distributed is a highly effective congestion control mechanism. Specifically, our experiments show that Vivace-Distributed always enters online learning phase in a sending rate that exceed the capacity of link. This means that Vivace-Distributed's online-learning phase only needs to adjust its sending rates to a stable point that utilizes all network resources, rather than probing for more bandwidth. This results in high utilization. The performance of sending rates demonstrates that Vivace-Distributed satisfies the criteria for a congestion control mechanism that is both highly efficient and stable.

7.3. Resilience to random loss

Using Emulab, we evaluate the throughput of different congestion control mechanisms with a single flow on the link with 100 Mbps bandwidth, 30 ms RTT, and varying random loss rates. We run a flow for 30 s (much longer than needed for convergence) ten times and take the average of ten runs as the throughput of each mechanism. We also evaluate the standard deviation of ten times simulations to access the stability of each mechanism. As depicted in Fig. 6, both Vivace and Vivace-Distributed utilize over 90% of the available bandwidth when the random loss rate is less than 3%; Vivace's average throughput remains nearly 60 Mbps when the random loss rate is 4%. However, as shown in Fig. 7, Vivace suffers from burst losses, which caused the average throughput to sometimes drop to as low as 1/10 of the link capacity. The utility function may sometimes be trapped at a local optimum due to random losses, resulting in low network bandwidth utilization. After that point, corresponding to the proof proposed in PCC-Vivace, when the random loss rate is 6%, the average throughput of Vivace can range from 6 Mbps to 63 Mbps, with most time utilizing only 1/10 of the bandwidth. In contrast, Vivace-Distributed's average throughput in the 30 s also decreases as the random loss rate increases. When the random loss rate is 4%, Vivace-Distributed 's average throughput drops to 88 Mbps. However, Vivace-Distributed's stability is much better, with a smaller standard deviation than that of Vivace. Conventional congestion control mechanisms have a small standard deviation, but their throughput is always less than 1 Mbps, which we think is the reason why their standard deviation is small. Compared with previous online-learning based mechanism Vivace, the newly introduced aggregative variable results in a larger utility gradient, which reduces the probability of being trapped in the local optimum. This improves the stability of the mechanism.

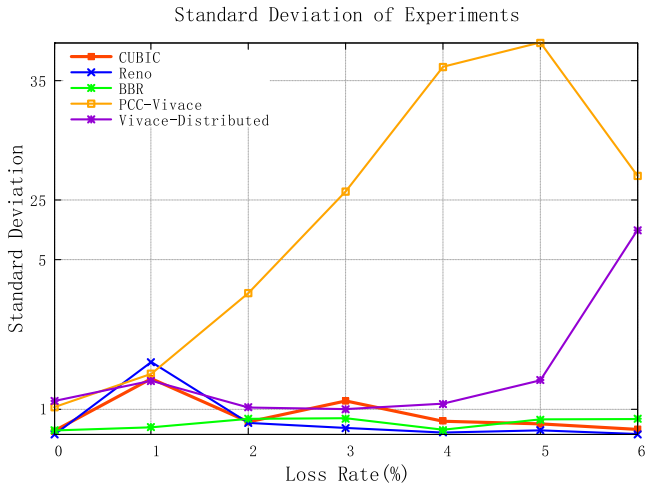


Fig. 7. Stability of throughput.

BBR's average throughput remains 90 Mbps until a 15% random loss; Vivace-Distributed can achieve the same random loss resilience by adjusting the parameter c. However, our experiments show that BBR's higher loss resilience induces comparable congestion loss with multiple competing flows, which we think is not reasonable for the network.

7.4. Convergence properties

In this part, we demonstrate that Vivace-Distributed improves the convergence speed by extending the duration of the slow start phase compared to existing mechanisms. We evaluate the convergence time and duration of the slow start for different mechanisms on a link with 100 Mbps bottleneck bandwidth, 30 ms RTT, and varying random loss rates.

7.4.1. Convergence speed

To evaluate the convergence time, we let a flow runs for 10 s, which is significantly longer than needed for convergence. The convergence time is defined as the time it takes from the flow's entry to the earliest time, after which the difference in throughput of two MIs is within ±25% of its ideal share of bandwidth.

Fig. 10 illustrates the convergence time of different congestion control mechanisms; when there is no random loss, Cubic and Reno take only about 1.5 s to reach the stable point. However, due to the "RTT Derivation" in the utility function, PCC-Vivace finds that it can reduce RTT by reducing its rate, causing it to back off even when there is no competition and taking nearly 4 s to reach the stable point. As shown in Fig. 5., Vivace-Distributed enters online learning phase with a sending rate much higher than the bandwidth, resulting in a higher convergence speed. This is due to the newly integrated variable, which delays the exit from the slow start phase. When there are random losses in the network, CUBIC and Reno remain taking about 3 s to the stable point. However, experiments show that their high convergence speed results from low bandwidth utilization. When there is random loss, only 5% bandwidth is utilized by Reno and CUBIC. In comparison, Vivace-Distributed takes about 4 s to the stable point but utilizes 85% of the network bandwidth. At the same time, compared with previous online-learning based mechanism Vivace, Vivace-Distributed achieve the same bandwidth utilization with a much quicker convergence time. Due to the aggregative variable acting as a reward in the utility function upon convergence, Vivace-Distributed has a longer slow start phase compared with Vivace and other mechanisms, which is shown in the following section.

7.4.2. Duration of slow start phase

Comparing the slow start phase with other phases, we observed that a sender takes network bandwidth more aggressively during slow start; Thus, our design mimics the way traditional congestion control behaves at the start stage, but instead of implementing a more aggressive slow-start algorithm, we aim to extend the duration of the slow start phase.

In CUBIC and Reno's slow start phase, the source sends twice of last time's sending packets every time an ACK is received. The slow start phase ends when the window reaches a certain level called the slow start threshold. When there is a loss, the slow start threshold is halved. However, due to non-congestion loss, the threshold is also halved even when there is no congestion loss. As a result, senders exist slow start phase with a low sending rate. As shown in Fig. 9(a), Fig. 9(b), and Fig. 9(c), when there is non-congestion loss, CUBIC and Reno's slow start phase lasts for 0.3 s, as the loss rate increases, it decreases to 0.2 s. In comparison, the utility-based mechanisms have a much longer slow start phase. Vivace's slow start lasts for 0.8 s when there is a 1% loss and 0.6 s when there is a 5% loss. Compared with Vivace, Vivace-Distributed's slow start lasts for nearly 0.9 s, and is not affected by random packet loss. Due to the aggregative variable, Vivace-Distributed is more certain whether the loss is loss-induced congestion or not; the utility value remains increasing for a longer time, resulting in a longer slow start phase.

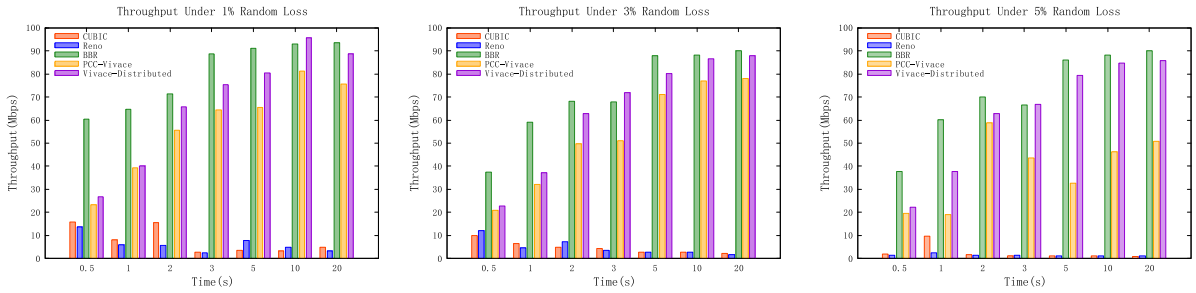
We do not evaluate BBR because BBR uses startup instead of the slow start, in which sending rates increase much faster than in slow start. This may result in a much lower TCP friendliness which we think is unsuitable for the real-world Internet.

As analyzed above, Vivace already has twice the duration of CUBIC and Reno's slow start phase to 0.6 s due to its utility-based architecture. Compared with Vivace, Vivace-Distributed further extends the duration of the slow start phase to 0.9 s. Although only 0.3 s is extended, the sending rate may be much higher, since the sending rate is double every time. We believe that extending the duration of slow start phase is more suitable as when more flows are competing for network resources, the duration of slow start can also be determined by the competition among flows. Thus, we believe Vivace-Distributed is a suitable solution to JointCloud environment.

7.5. Throughput of different flows

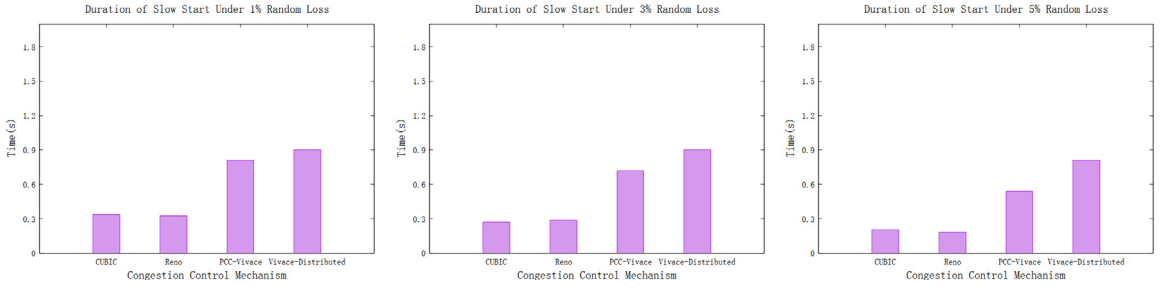
To evaluate the performance in JointCloud environment, we use Emulab to set up an emulated link with 100 Mbps bandwidth, 30 ms RTT, 75 KB buffer, and 1%, 3%, 5% random loss rate. To emulate different JointCloud services (i.e., data transmission, web search), we set up different TCP connections lasting for 0.5, 1 s, 2 s, 5 s, 10 s, and 20 s. As shown in Fig. 8(a), Fig. 8(b), and Fig. 8(c), when there is random loss in the network, CUBIC and Reno suffer from random loss. Conventional rule-based congestion control mechanisms suffer from notoriously bad performance due to non-congestion packet loss. The throughput of the conventional rule-based mechanism remains less than 2 Mbps, and only 1/50 of the bandwidth is utilized. The utility-based architecture shows the advantage of enduring random packet loss, which gains a 20–50× throughput over traditional rule-based CC mechanisms.

Compared with PCC-Vivace, the proposed Vivace-Distributed achieves a higher utilization of network resources, especially for short flows. Results from Fig. 8(a), Fig. 8(b), and Fig. 8(c), show that when a short flow lasts for 1-10 s, Vivace-Distributed achieves a 30% higher throughput compared to Vivace. This is mainly due to Vivace-Distributed existing the slow start phase with a higher sending rate. Furthermore, in the online-learning phase, aggregative variable acts as a newly integrated reward,



(a) 1% Random Loss (b) 3% Random Loss (c) 5% Random Loss

Fig. 8. Throughput of different flows.



(a) No Random Loss (b) 1% Random Loss (c) 3% Random Loss

Fig. 9. Duration of slow start.

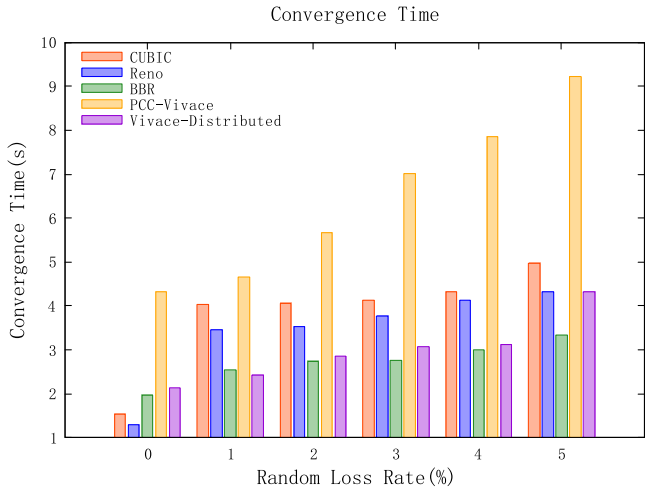


Fig. 10. Temporal behavior of convergence.

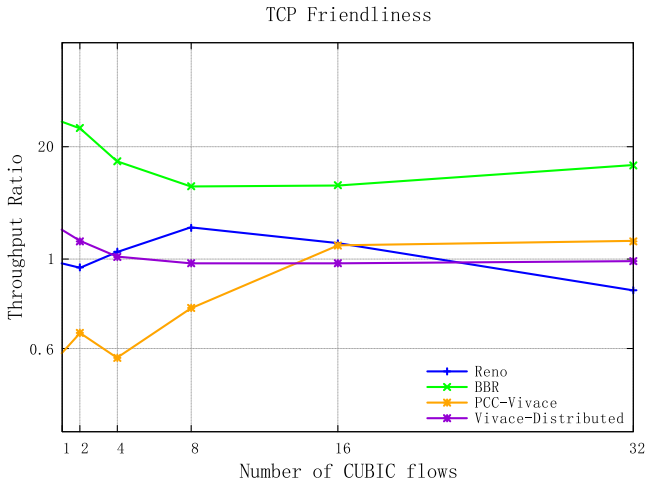


Fig. 11. TCP Friendliness.

which leads to a quicker convergence to the Nash Equilibrium. When a flow lasts for more than 10 s, Vivace-Distributed achieves a 12% higher throughput at 1% random loss rate and 2× higher performance at 5% random loss rate. When a flow lasts for more than 10 s, 10 s is apparently longer than converge time, which means that both Vivace and Vivace have achieved the Nash Equilibrium and the throughout is mainly affected by non-congestion loss. However, due to the new reward in Vivace-Distributed's utility function, Vivace-Distributed has a higher recovery speed from non-congestion loss and achieves a better performance.

7.6. TCP friendliness

To evaluate the TCP friendliness of different mechanisms, we use a bottleneck link with 30 ms RTT, 100 Mbps bandwidth, and 75 KB buffer. One flow utilizing a new mechanism (PCC-Vivace, Vivace-Distributed, or BBR) shares bandwidth with an increasing number of CUBIC flows. We choose CUBIC as the primary flow since it is the default congestion control mechanism for the Linux kernel, which we believe is the most widely used in today's network. Under this circumstance, all competing flows share a bandwidth of 100 Mbps, and as the number of flows increases, total bandwidth and buffer size are not increased. We aim to test whether different mechanisms are able to back off and release their network resources for newly established connections. In real-world Internet scenarios, the bandwidth remains unchanged for a long time. Thus, if a flow utilizes too many network resources for a long time, congestion-induced loss will occur, leading to a bad performance for competing flows. Fig. 11 shows the ratio between the throughput of the flow utilizing the

new mechanism, and the average throughput of the CUBIC flows. A ratio of 1 indicates perfect friendliness, indicating that network resources are shared fairly among different flows. A large ratio indicates aggressiveness towards CUBIC.

As shown in Fig. 11, when there are only a few CUBIC flows, Vivace backs off, even though the queue is not always full, as Vivace finds that it can reduce the RTT by reducing its rate. However, as more TCP connections are established, Vivace achieves perfect friendliness. Vivace is too conservative when the number of competing flows is small, which is inappropriate for many cloud services. Compared to Vivace, Vivace-Distributed works more aggressively when there is no competition in JointCloud. Due to the aggregative variable, Vivace-Distributed takes under-utilized bandwidth actively. As more and more competing flows are established, Vivace-Distributed also achieves perfect friendliness. As the number of flows increases, the aggregative variable in Vivace-Distributed prevents it from taking too many network resources. As a result, Vivace-Distributed also achieves perfect friendliness. The ratio is slightly less than 1 since non-congestion-induced latency jitters often occur.

The throughput tests on different emulated cloud services indicate that Vivace-Distributed outperforms Vivace, especially on short flows. The throughput ratio also demonstrates that Vivace-Distributed not only takes over under-utilized bandwidth but also frees up capacity for newly established connections. The results show that Vivace-Distributed is more aggressive when there are no competing flows, but it also achieves perfect friendliness as the number of flows increases. In summary, Vivace-Distributed is able to balance its aggressiveness with fairness, which is crucial for cloud services that require efficient resource utilization while maintaining fairness among competing flows. It meets JointCloud's requirements for a high efficient and fair congestion control mechanism.

8. Conclusion

We proposed Vivace-Distributed, a novel congestion control mechanism designed for JointCloud environments. Vivace-Distributed leverages distributed online learning with an aggregative variable that represents the decisions of all senders, working in a distributed manner. Our theoretical analysis establishes that Vivace-Distributed can always achieve a Nash Equilibrium. Extensive experiments further demonstrate that Vivace-Distributed achieves a stable point. Due to the aggregative variable, Vivace-Distributed has a higher convergence speed and achieves a higher throughput especially for short flows through extending the duration of slow start phase. Overall, Vivace-Distributed provides valuable insights into the development of congestion control mechanisms for JointCloud environments. In the future, We plan to integrate Vivace-Distributed into emulators such as Pantheon [39] and production systems such as QUIC [40].

CRediT authorship contribution statement

Jianzhi Shi: Conceptualization, Methodology, Software, Writing – original draft. **Bo Yi:** Writing – review & editing, Visualization, Supervision, Funding acquisition. **Xingwei Wang:** Writing – review & editing, Visualization, Supervision, Funding acquisition. **Min Huang:** Writing – review & editing, Visualization, Supervision, Funding acquisition. **Peichen Li:** Data curation, Investigation. **Chao Zeng:** Validation, Resources. **Keqin Li:** Writing – review & editing, Visualization, Supervision, Funding acquisition.

Declaration of competing interest

No potential conflict of interest was reported by the authors.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China (2022YFB4500800), and the National Natural Science Foundation of China (62032013, 92267206).

References

[1] Shangquan Gao, Economic globalization: trends, risks and risk prevention, in: Economic & Social Affairs, CDP Backround Paper, Vol. 1, 2000.
[2] H. Wang, P. Shi, Y. Zhang, Jointcloud: A cross-cloud cooperation architecture for integrated internet service customization, in: 2017 IEEE 37th International Conference on Distributed Computing Systems, ICDCS, IEEE, 2017, pp. 1846–1855.
[3] T. Zhang, S. Mao, Machine learning for end-to-end congestion control, IEEE Commun. Mag. 58 (6) (2020) 52–57.
[4] M. Akbari, B. Gharesifard, T. Linder, Distributed online convex optimization on time-varying directed graphs, IEEE Trans. Control Netw. Syst. 4 (3) (2015) 417–428.
[5] X. Li, X. Yi, L. Xie, Distributed online convex optimization with an aggregative variable, IEEE Trans. Control Netw. Syst. 9 (1) (2021) 438–449.
[6] M. Wu, Z. Mi, Y. Xia, A survey on serverless computing and its implications for jointcloud computing, in: 2020 IEEE International Conference on Joint Cloud Computing, IEEE, 2020, pp. 94–101.
[7] M. Guzek, A. Gniewek, P. Bouvry, et al., Cloud brokering: Current practices and upcoming challenges, IEEE Cloud Comput. 2 (2) (2015) 40–47.
[8] H. Yang, J. Yuan, H. Yao, et al., Blockchain-based hierarchical trust networking for JointCloud, IEEE Internet Things J. 7 (3) (2019) 1667–1677.
[9] P.B. Velloso, D.C. Morales, T.M.T. Nguyen, et al., BASICS: A multi-blockchain approach for securing VM migration in joint-cloud systems, in: 2023 IEEE 20th Consumer Communications & Networking Conference, CCNC, IEEE, 2023, pp. 523–528.
[10] Z. Huang, Z. Mi, Z. Hua, HCloud: A trusted JointCloud serverless platform for IoT systems with blockchain, China Commun. 17 (9) (2020) 1–10.
[11] H. Yang, J. Yuan, H. Yao, et al., Blockchain-based hierarchical trust networking for JointCloud, IEEE Internet Things J. 7 (3) (2019) 1667–1677.
[12] B. Yin, L. Mei, Z. Jiang, et al., Joint cloud collaboration mechanism between vehicle clouds based on blockchain, in: 2019 IEEE International Conference on Service-Oriented System Engineering, SOSE, IEEE, 2019, pp. 227–2275.
[13] Microsoft hybrid cloud. https://www.microsoft.com/en-us/cloud/platform/hybrid-cloud.
[14] VMware ahybrid cloud solutions. http://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/brochure/vmware-hybrid-cloud-brochure.pdf.
[15] Virtual Extensible Local Area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks, in: IETF RFC 7348, 2014, Aug.
[16] K. Yuan, F. Wang, Z. Marszalek, A delay-tolerant data congestion avoidance algorithm for enterprise cloud system based on modular computing, Mob. Netw. Appl. (2022) 1–11.
[17] R. Karthikeyan, V. Balamurugan, R. Cyriac, et al., COSCO2: AI-augmented evolutionary algorithm based workload prediction framework for sustainable cloud data centers, Trans. Emerg. Telecommun. Technol. 34 (1) (2023) e4652.
[18] R. Al-Saadi, G. Armitage, J. But, et al., A survey of delay-based and hybrid TCP congestion control algorithms, IEEE Commun. Surv. Tutor. 21 (4) (2019) 3609–3638.
[19] N. Cardwell, Y. Cheng, C.S. Gunn, et al., Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time, Queue 14 (5) (2016) 20–53.
[20] S. Ha, I. Rhee, L. Xu, CUBIC: a new TCP-friendly high-speed TCP variant, ACM SIGOPS Oper. Syst. Rev. 42 (5) (2008) 64–74.
[21] B. Braden, D. Clark, J. Crowcroft, et al., RFC2309: Recommendations on queue management and congestion avoidance in the internet, 1998.
[22] S. Floyd, T. Henderson, A. Gurtov, Rfc3782: The newreno modification to tcp's fast recovery algorithm, 2004.
[23] L.S. Brakmo, S.W. O'Malley, L.L. Peterson, TCP Vegas: New techniques for congestion detection and avoidance, in: Proceedings of the Conference on Communications Architectures, Protocols and Applications, 1994, pp. 24–35.
[24] K. Winstein, H. Balakrishnan, Tcp ex machina: Computer-generated congestion control, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 123–134.

[25] M. Dong, Q. Li, D. Zarchy, et al., PCC: Re-architecting congestion control for consistent high performance, in: 12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, 2015, pp. 395–408.
[26] M. Dong, T. Meng, D. Zarchy, et al., PCC vivace: Online-learning congestion control, in: 15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 18, 2018, pp. 343–356.
[27] V. Badarla, C.S.R. Murthy, Learning-tcp: A stochastic approach for efficient update in tcp congestion window in ad hoc wireless networks, J. Parallel Distrib. Comput. 71 (6) (2011) 863–878.
[28] Q. He, Y. Wang, X. Wang, et al., Routing optimization with deep reinforcement learning in knowledge defined networking, IEEE Trans. Mob. Comput. (2023).
[29] A. Sivaraman, K. Winstein, P. Thaker, et al., An experimental study of the learnability of congestion control, ACM SIGCOMM Comput. Commun. Rev. 44 (4) (2014) 479–490.
[30] M. Schapira, K. Winstein, Congestion-control throwdown, in: Proceedings of the 16th ACM Workshop on Hot Topics in Networks, 2017, pp. 122–128.
[31] R. Boutaba, M.A. Salahuddin, N. Limam, et al., A comprehensive survey on machine learning for networking: evolution, applications and research opportunities, J. Internet Serv. Appl. 9 (1) (2018) 1–99.
[32] E. Even-Dar, Y. Mansour, U. Nadav, On the convergence of regret minimization dynamics in concave games, in: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, 2009, pp. 523–532.
[33] M. Zinkevich, Online convex programming and generalized infinitesimal gradient ascent, in: Proceedings of the 20th International Conference on Machine Learning, icml-03, 2003, pp. 928–936.
[34] F. Dobrian, V. Sekar, A. Awan, et al., Understanding the impact of video quality on user engagement, ACM SIGCOMM Comput. Commun. Rev. 41 (4) (2011) 362–373.
[35] A. Jafari, A. Greenwald, D. Gondek, et al., On no-regret learning, fictitious play, and nash equilibrium, in: ICML, Vol. 1, 2001, pp. 226–233.
[36] R. Xie, X. Jia, K. Wu, Adaptive online decision method for initial congestion window in 5G mobile edge computing using deep reinforcement learning, IEEE J. Sel. Areas Commun. 38 (2) (2019) 389–403.
[37] Linux net-next. https://kernel.googlesource.com/pub/scm/linux/kernel/git/dave/net-next.git/+/v4.10.
[38] B. White, J. Lepreau, L. Stoller, et al., An integrated experimental environment for distributed systems and networks, Oper. Syst. Rev. 36 (SI) (2002) 255–270.
[39] F.Y. Yan, J. Ma, G.D. Hill, et al., Pantheon: the training ground for internet congestion-control research, in: 2018 USENIX Annual Technical Conference, USENIXATC 18, 2018, pp. 731–743.
[40] A. Langley, A. Riddoch, A. Wilk, et al., The quic transport protocol: Design and internet-scale deployment, in: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, 2017, pp. 183–196.

Jianzhi Shi received the BS degree in computer science and engineering from Northeastern University of China in 2022. He is currently working towards Ph.D. degree in computer science and engineering, Northeastern University of China. His current research interests include cloud computing, service computing, JointCloud, etc.

Bo Yi (Member,IEEE) is currently a lecturer of computer science and engineering with the Northeastern University of China. He has authored and co-authored more than 20 journal and conference articles on IEEE Transactions on Cloud Computing, IEEE Communications Letter, IEEE Access, Computer Networks, etc. He is currently the reviewer of IEEE Communications Survey & Tutorial, Communications Letter, Computer Networks, Journal of Network and Computer Applications, etc. His research interests include service computing, routing, virtualization, cloud computing in SDN, NFV, DetNet, etc.

Xingwei Wang received the BS, MS, and Ph.D. degrees in computer science from the Northeastern University, Shenyang, China, in 1989, 1992, and 1998, respectively. He is currently a professor with the College of Computer Science and Engineering, Northeastern University, Shenyang, China. His research interests include cloud computing and future Internet, etc. He has published more than 100 journal articles, books and book chapters, and refereed conference papers. He has received several best paper awards.

Min Huang received the BS degree in automatic instrument, the MS degree in systems engineering, and the Ph.D. degree in control theory from the Northeastern University, Shenyang, China, in 1990, 1993, and 1999, respectively.She is currently a professor with the College of Information Science and Engineering, Northeastern University, Shenyang, China.Her research interests include modeling and optimization for logistics and supply chain system, etc. She has published more than 100 journal articles, books, and refereed conference papers.

Peichen Li received a B.S. degree in Software College, Northeastern University, Shenyang, 2019. Currently, he is pursuing a Ph.D. in the Department of Computer Science and Engineering, Northeastern University. His research interests include intelligent routing and deep learning.

Chao Zeng received the MS degree in Software College, Northeastern University, Shenyang, 2022. Currently, he is pursuing a Ph.D. in the Department of Computer Science and Engineering, Northeastern University. His research interests include JointCloud computing and federated learning.

Keqin Li (Fellow,IEEE) is a SUNY Distinguished Professor of Computer Science with the State University of New York. He is also a National Distinguished Professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber–physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU–GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored over 890 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds nearly 70 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top 5 most influential scientists in parallel and distributed computing in terms of both single-year impact and career-long impact based on a composite indicator of Scopus citation database. He has chaired many international conferences. He is currently an associate editor of the ACM Computing Surveys and the CCF Transactions on High Performance Computing. He has served on the editorial boards of the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Computers, the IEEE Transactions on Cloud Computing, the IEEE Transactions on Services Computing, and the IEEE Transactions on Sustainable Computing. He is an AAAS Fellow, an IEEE Fellow, and an AAIA Fellow. He is also a Member of Academia Europaea (Academician of the Academy of Europe).