Universidade Federal de Pernambuco
Centro de Informática

Pós-Graduação em Ciência da Computação

An Experimental Analysis of TCP Congestion Control Algorithms Within Virtualized Environments

Pedro Rafael Ximenes do Carmo

Dissertação de Mestrado

Recife
2024

Universidade Federal de Pernambuco
Centro de Informática

Pedro Rafael Ximenes do Carmo

An Experimental Analysis of TCP Congestion Control Algorithms Within Virtualized Environments

Trabalho apresentado ao Programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Prof. Djamel F. H. Sadok*

Recife
2024

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Carmo, Pedro Rafael Ximenes do.
An experimental analysis of TCP congestion control algorithms within virtualized environments / Pedro Rafael Ximenes do Carmo. - Recife, 2024.
118f.: il.

Dissertação (Mestrado), Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós-Graduação em Ciência da Computação, 2024.
Orientação: Djamel Fawzi Hadj Sadok.

1. Virtualization; 2. TCP; 3. Network Congestion. I. Sadok, Djamel Fawzi Hadj. II. Título.

UFPE-Biblioteca Central CDD 004.6

Pedro Rafael Ximenes do Carmo

“An Experimental Analysis of TCP Congestion Control Algorithms Within Virtualized Environments”

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação. Área de Concentração: Redes de Computadores e Sistemas Distribuídos.

Aprovado em: 29/07/2024.

BANCA EXAMINADORA

Prof. Dr. Renato Mariz de Moraes
Centro de Informática / UFPE

Prof. Dr. Eduardo James Pereira Souto
Instituto de Computação / UFAM

Prof. Dr. Djamel Fawzi Hadj Sadok
Centro de Informática / UFPE
(**orientador**)

Acknowledgements

Agradeço a todos que, de alguma forma, contribuíram para a realização desta dissertação. Em especial os meus pais, Pedro e Ana, à minha namorada, Dayane, aos meus avós, tios e padrinhos que sempre sonharam comigo, aos Professores Djamel Sadok e Judith Kelner, aos meus colegas de pesquisa, Assis, Eduardo e Diego, e todos os outros colegas do Grupo de Pesquisa em Redes e Telecomunicações (GPRT). Todo o apoio, incentivo e colaboração foram essenciais para a conclusão deste trabalho. Meu sincero muito obrigado a cada um de vocês.

They say the universe is expanding. That should help with the traffic.
—STEVEN WRIGHT

Abstract

In the growing landscape of virtualized networks, the performance of TCP congestion control algorithms remains a critical factor in ensuring efficient data transmission. This dissertation presents a comparative and experimental analysis of four prominent TCP congestion control algorithms – Vegas, CUBIC, BBRv2, and DCTCP – in virtualized environments. Motivated by the need to understand how these algorithms work in virtualized environments, this study investigates their behavior in various scenarios with varying network conditions, including baseline performance, under basic network failures, and in competitive scenarios. This study differs from others found in the literature by evaluating virtualization scenarios and using a physical testbed environment instead of simulations to evaluate the performance of TCP congestion control algorithms. The testbed consists of dedicated physical servers and network devices configured to emulate various network conditions. This configuration enables precise control and reproducibility of experiments, providing accurate measurements of key evaluation metrics: sending rate, throughput, throughput fairness, round trip time (RTT), and retransmission rates. The findings indicate that, in virtualized environments, algorithms such as Vegas, CUBIC, DCTCP, and BBRv2 exhibit unique performance characteristics that affect network efficiency and reliability. Factors such as resource sharing and overhead between virtual machines impact the algorithm's performance. Delay-based algorithms such as Vegas are more affected by virtualization-induced latency. At the same time, CUBIC's window growth strategy can lead to suboptimal performance due to increased queuing delays in virtual switches. BBRv2's balanced approach is well suited to the dynamic conditions imposed by virtualization but can be affected by additional processing overhead and variable latency. The study concludes that no algorithm universally outperforms the others in all scenarios. Instead, the choice of congestion control algorithm should depend on the context, considering specific network conditions and performance requirements. This dissertation contributes to understanding the dynamics of TCP congestion control in virtualized environments, offering insights that can guide the selection and optimization of these algorithms to improve network performance. Based on the findings, network administrators managing virtualized environments should select TCP congestion control algorithms according to specific operational needs. Vegas is ideal for minimizing latency, CUBIC and DCTCP for maximizing throughput, and BBRv2 for maintaining fairness and adaptability in dynamic network conditions. Furthermore, the

study reveals that the virtualization context introduces an additional layer of complexity when deploying these algorithms in cloud-based scenarios. This critical distinction highlights the need to account for the unique challenges posed by virtualization when evaluating and optimizing TCP performance in modern data center environments.

Keywords: Virtualization, Network Congestion, Congestion Control Algorithms, Cloud Computing, Virtual Machines (VMs)

Resumo

No cenário crescente das redes virtualizadas, o desempenho dos algoritmos de controle de congestionamento TCP continua sendo um fator crítico para garantir uma transmissão de dados eficiente. Esta dissertação apresenta uma análise comparativa e experimental de quatro algoritmos proeminentes de controle de congestionamento TCP – Vegas, CUBIC, BBRv2 e DCTCP – em ambientes virtualizados. Motivado pela necessidade de entender como esses algoritmos funcionam em ambientes virtualizados, este estudo investiga seu comportamento em vários cenários variando as condições de rede. Isso inclui desempenho *baseline*, sob falhas básicas de rede e em cenários competitivos. Este estudo se diferencia dos demais encontrados na literatura ao avaliar cenários de virtualização e utilizar um ambiente de teste físico em vez de simulações para avaliar o desempenho de algoritmos de controle de congestionamento TCP. A infraestrutura de teste consiste em servidores físicos dedicados e dispositivos de rede configurados para emular uma variedade de condições de rede. Essa configuração permite controle preciso e reprodutibilidade de experimentos, fornecendo medições precisas das principais métricas de avaliação: taxa de envio, taxa de transferência, índice de justiça de *throughput*, tempo de ida e volta (RTT) e taxas de retransmissão. As descobertas indicam que, em ambientes virtualizados, algoritmos como Vegas, CUBIC, DCTCP e BBRv2 exibem características de desempenho unicas que afetam a eficiência e a confiabilidade da rede. Fatores como compartilhamento de recursos e sobrecarga entre máquinas virtuais impactam o desempenho do algoritmo. Algoritmos baseados em atraso, como Vegas, são mais afetados pela latência induzida pela virtualização, enquanto a estratégia de crescimento de janela do CUBIC pode levar a um desempenho abaixo do ideal devido ao aumento dos atrasos nas filas em switches virtuais. A abordagem equilibrada do BBRv2 é adequada às condições dinâmicas impostas pela virtualização, mas pode ser afetada pela sobrecarga adicional de processamento e pela latência variável. O estudo conclui que nenhum algoritmo supera universalmente os outros em todos os cenários. Em vez disso, a escolha do algoritmo de controle de congestionamento deve depender do contexto, considerando condições específicas da rede e requisitos de desempenho. Esta dissertação contribui para a compreensão da dinâmica de controle de congestionamento TCP em ambientes virtualizados, oferecendo *insights* que podem orientar a seleção e otimização desses algoritmos para melhorar o desempenho da rede. Com base nas descobertas, os administradores de rede que gerenciam ambientes virtualizados devem se-

lecionar algoritmos de controle de congestionamento TCP de acordo com necessidades operacionais específicas. Vegas é ideal para minimizar a latência, CUBIC e DCTCP para maximizar o *throughput* e BBRv2 para manter a imparcialidade e adaptabilidade em condições de rede dinâmicas. Além disso, o estudo revela que o contexto de virtualização introduz uma camada adicional de complexidade ao implantar esses algoritmos em cenários baseados em nuvem. Essa distinção crítica destaca a necessidade de levar em conta os desafios exclusivos impostos pela virtualização ao avaliar e otimizar o desempenho do TCP em ambientes modernos de data center.

Palavras-chave: Virtualização, Congestionamento de Rede, Algoritmos de Controle de Congestionamento, Computação em Nuvem, Máquinas Virtuais (VMs)

Contents

List of Figures

List of Tables

Acronyms

ACK Acknowledgement.

BBR Bottleneck Bandwidth and Round-trip propagation time.

BBR2 Bottleneck Bandwidth and Round-trip propagation time version 2.

BDP Bandwidth-Delay Product.

BIC Binary Increase Congestion Control.

bps Bits per second.

BtlBw Bottleneck Bandwidth.

CBR Constant Bit Rate.

CTCP Compound TCP.

CUBIC Cubic Congestion Control.

Cwnd Congestion Window.

DCN Data Center Network.

DCTCP Data Center TCP.

ECN Explicit Congestion Notification.

FAST Fast Active Queue Management Scalable TCP.

Gbps Gigabits per second.

HS-TCP High-Speed TCP.

IaaS Infrastructure as a Service.

IT Information Technology.

KBytes Kilobytes.

KVM Kernel-based Virtual Machine.

Mbps Megabits per second.

ms Milliseconds.

MTU Maximum Transmission Unit.

NETEM Network Emulator.

NFV Network Function Virtualization.

NIC Network Interface Card.

NS2 Network Simulator 2.

NS3 Network Simulator 3.

OS Operating System.

PaaS Platform as a Service.

PCC Performance Oriented Congestion Control.

QoS Quality of Service.

RTprop Round-Trip Propagation Time.

RTT Round-Trip Time.

SaaS Software as a Service.

SDN Software-Defined Networking.

SR-IOV Single Root Input/Output Virtualization.

SYN Synchronize.

SYN-ACK Synchronize-Acknowledge.

TC Traffic Control.

TCP Transmission Control Protocol.

TCP/IP Transmission Control Protocol/Internet Protocol.

TLS Transport Layer Security.

txqueuelen Transmit Queue Length.

UDP User Datagram Protocol.

vCC Virtualized Congestion Control.

VEGAS Vegas Congestion Control Algorithm.

VM Virtual Machine.

vMCC Virtualized Multipath Congestion Control.

CHAPTER 1

Introduction

In the rapidly evolving realm of the internet, virtualized environments have become pivotal in enhancing operational efficiency. They play a crucial role in resource optimization, significantly altering the management and execution of Information Technology (IT) operations [ACA21]. The IT field encompasses the use of computers, networks, storage systems, and other hardware and software for processing, storing, and exchanging data. This advancement facilitates a more agile and scalable infrastructure, which is critical for adapting to rapidly changing technological demands.

One of the benefits of virtualization technology is that it allows multiple virtual instances to share a single physical hardware, significantly reducing the need to invest in extensive physical infrastructures. This leads to substantial Cost Reduction, as it requires fewer hardware purchases and maintenance [RAZ20]. Furthermore, virtualization supports a seamless, prompt implementation of applications running on different operating systems, empowering companies to adjust efficiently to evolving industry trends. Centralized Management in virtualized environments simplifies administrative tasks, ensuring smoother operations and enhanced security. Lastly, unlike physical hosts, Virtual Machines (VMs) are flexible software components that may be migrated and replicated on demand. For example, VM replication supports zero downtime [CFH+05] software upgrade operation, whereas VM migration may be used to achieve energy saving, load balancing, and high availability services [Bel13].

Among the various virtualization solutions, Kernel-based Virtual Machine (KVM) technology stands out for its efficiency and scalability, which is why it has been selected for experimental analysis in this dissertation. KVM transforms the Linux kernel into a hypervisor, enabling the operation of multiple virtual instances on a single physical hardware [Got11]. In addition to cost savings, KVM capitalizes on the robust security and performance features of the Linux kernel. It supports a wide range of guest operating systems, facilitating the implementation of applications across diverse environments. This flexibility empowers companies to adjust efficiently to evolving industry trends. Lastly, integrating KVM with existing Linux system management tools facilitates centralized management of the virtualized environment, simplifies administrative tasks, ensures smoother operations, and enhances security.

Given that our concern is with packet processing, focus is given to Transmission Control Protocol (TCP) flows and their delay at the data plane. TCP is fundamental

in Internet architecture and ensures reliable and ordered delivery of data transmission packets across networks [Pos81]. Its Congestion Control mechanism is vital in maintaining network stability and efficiency. TCP's Connection-Oriented nature guarantees that data packets are delivered in order and without loss, making it indispensable for most high-level applications such as Web and e-mail. Furthermore, its Standardization and Interoperability ensure consistent performance across diverse network infrastructures.

In virtualized environments, the dynamics of TCP congestion control assume new and complex aspects. These dynamics are poorly understood and have notable gaps in existing literature, highlighting the pressing need for further research [TTS15]. TCP's congestion control algorithms face unique challenges in these environments due to the nature of the virtual network interfaces and shared physical resources. This dissertation delves into an experimental analysis of these algorithms within such environments, aiming to uncover insights that could lead to more efficient network traffic management [NG23].

This research empirically analyzes the performance of various TCP congestion control algorithms within a virtualized context, focusing on Vegas, CUBIC, BBR2, and DCTCP. These algorithms represent a broad spectrum of approaches to congestion control, including Vegas's proactive packet loss avoidance based on anticipated congestion signals, CUBIC's scalability and robustness in diverse network conditions, BBR2's approach aiming at maximizing bandwidth while minimizing latency, DCTCP's ability to fine-tune its response to congestion signals in data center environments. By evaluating these algorithms under the unique constraints and opportunities presented by virtualized environments, the study aims to highlight their efficiencies, limitations, and the impact of virtualization on their operational dynamics. Through a detailed experimental framework that includes baseline scenario analysis, network failure simulations, and multiple algorithm competitions for network and processing resources, this dissertation seeks to provide a comprehensive understanding of how TCP congestion control algorithms perform and interact in virtualized settings. The outcomes of this research are expected to offer valuable insights for network administrators and system designers in optimizing TCP congestion control within virtualized infrastructures, thereby enhancing overall network performance and reliability.

1.1 Research Questions

To guide the empirical investigation and ensure a focused analysis, the following research questions have been formulated:

1. **How do TCP congestion control algorithms perform in a virtualized environment?** This question seeks to understand the operational dynamics of the adopted

algorithms when faced with the unique processing overhead of virtualization due to shared resources and the use of virtual network interfaces.

2. **How do network conditions influence the behavior of TCP congestion control algorithms in virtualized environments?** This question intends to evaluate the adaptability and robustness of each congestion control algorithm under various stress conditions and network configurations.

3. **What recommendations can be made for network administrators and system designers in selecting and optimizing TCP congestion control algorithms for virtualized environments?** Based on the findings, this question aims to offer actionable insights and guidelines to enhance network performance and communication reliability in virtualized infrastructures.

Addressing these research questions will provide a comprehensive understanding of the performance of TCP congestion control algorithms in virtualized settings, offering valuable insights for the design and management of efficient and reliable networks.

1.2 Objectives

The primary objective of this dissertation is to conduct an in-depth experimental analysis of TCP congestion control algorithms within virtualized environments. This investigation aims to understand the interplay between of the algorithms' complex dynamics and performance metrics when deployed in such settings. The study sets forth the following specific objectives to accomplish these goals:

1. **To Assess the Performance of Selected TCP Congestion Control Algorithms**: This involves a detailed examination of BBR2, CUBIC, DCTCP, and VEGAS algorithms designed to understand their behavior and effectiveness in managing network congestion within virtualized environments.

2. **To Investigate the Effectiveness of Algorithms under Various Network Conditions**: This includes scenarios of baseline network operations, simulated network failures, and multiple algorithm deployments to mimic real-world network variability and stress conditions.

3. **To Measure Performance Using Key Evaluation Metrics**: Employing metrics such as throughput, sending rate, and end-to-end latency in the form of Round-Trip Time (RTT), throughput fairness, and retransmission rates to quantitatively assess the performance of each TCP congestion control algorithm.

4. **To Provide a Comparative Analysis of Algorithms**: Offering insights into each algorithm's comparative strengths and weaknesses, facilitating a better understanding of their suitability for different virtualized network scenarios.

5. **To Propose Recommendations for Optimal Algorithm Selection**: Based on the findings, suggest guidelines for network administrators and system designers on selecting appropriate TCP congestion control algorithms for specific virtualized environments and applications.

6. **To Contribute to the Body of Knowledge**: Enhance the existing literature on TCP congestion control by providing empirical evidence and analysis specific to virtualized environments, thereby addressing a significant gap in current research.

By accomplishing these objectives, this dissertation will build valuable insights and guidelines that could significantly improve the design, management, and optimization of TCP congestion control mechanisms in virtualized networking environments.

1.3 Work Organization

The dissertation is structured to provide an overview of related works (Section 2) and a comprehensive background (Section 3), followed by a detailed methodology (Section 4), experimental results (Section 5), and a discussion of the findings (Section 6). Finally, the dissertation concludes with limitations, future research, and final thoughts (Section 7). The ultimate goal is to contribute to optimizing TCP performance in virtualized environments, a key area in network management.

CHAPTER 2

Related Works

In this chapter, we survey the current state of the art in performance evaluation of TCP congestion control algorithms. The examined works contribute significantly to this dissertation by providing insights into the design of the experiments and methodologies adopted or as a theoretical reference to understand the nuances and behavior of these congestion control algorithms under specific conditions. Through this survey, we established a foundation for the proposed investigation, identifying gaps in the existing literature and outlining the path for this research's original contribution to the field of study. We have divided the chapter into two main sections to address these facets. These sections dissect the landscape through two distinct lenses: practical evaluation and experimentation of these algorithms, as well as theoretical discussions, surveys, and proposals for new algorithms. This dual approach furnishes the comprehensive background necessary for the proposed investigation and uncovers gaps within the existing literature, thereby delineating the trajectory for this research's novel contributions to the field.

2.1 Practical Evaluations and Experiments of TCP Congestion Control

This section delves into papers concerned with practical evaluations and experiments with TCP congestion control algorithms. It highlights the significance of empirical studies in shedding light on the algorithms' real-world performance and behavior. Such information offers a tangible basis for defining the algorithms that best fit the different contemporary network demands. The result of this analysis serves as a comparative basis with the work proposed in this dissertation, showing the gaps in the current literature and how this dissertation addresses them.

The study by Turkovic et al. [TKU19] investigates congestion control while considering the performance and interactions of various congestion control algorithms of the TCP and QUIC protocols, with a greater emphasis put on TCP. Given the role of congestion control in maintaining the efficiency and stability of network traffic, the research is motivated by the need to understand how different algorithms – categorized as loss-based, delay-based, and hybrid – deal with resource allocation challenges and network performance in shared environments. The first objective of the research is to

perform a comparative analysis of these algorithms to identify the way they manage network resources, respond to congestion signals, and impact overall network throughput and latency. Secondly, the study aims to explore the interactions between these algorithms when they coexist within the same network infrastructure, an aspect previously little explored in the literature. The ultimate goal is to discover insights that can guide the development of more effective but also fair congestion control strategies, especially in increasingly complex and demanding network scenarios.

The researchers built a controlled test environment to emulate various network conditions, enabling precise analysis of how different congestion control algorithms work and interact. This configuration involved implementing a network topology that could mimic real-world Internet paths, incorporating scenarios with varying RTTs and network bottlenecks. The network topology used in the study is known as “dumbbell" topology. This configuration is widely used in network research to simulate how different data streams interact and compete for bandwidth at a central congestion point. In the topology used by the authors, two sets of hosts, clients, and servers are connected through two central routers that are in turn connected, representing the network bottleneck. Each side of the “dumbbell" simulates a distinct local network; on the client side, multiple Hosts generate traffic that traverses the central congestion point to reach server hosts on the other side, representing a common congestion point – a scenario often encountered in communications networks. This framework allows researchers to directly observe the impact of different congestion control algorithms under controlled conditions, including the response to RTT variations. The methodology involves classifying congestion control algorithms into three main groups – loss-based, delay-based, and hybrid – and evaluating their performance, including throughput, fairness, and RTT-fairness metrics. The researchers used a comprehensive set of tools and techniques for this analysis, including *IPERF3* for network traffic generation, *TSHARK* for packet capture and analysis, and *TC* and *NETEM* of Linux for emulating network conditions (for example, introducing artificial latency and limiting the bandwidth available on the bottleneck link). As previously mentioned, the evaluation criteria also include transfer rate and throughput to measure the efficiency of data transfer under different congestion control mechanisms; fairness, to evaluate the equitable distribution of bandwidth between different flows, especially in situations where different algorithms compete for resources; and RTT fairness, which examines the impact of RTT variations on network resource allocation, a common phenomenon in real-world networks.

The experiments carried out by Turkovic et al. were organized into distinct phases, each focused on specific aspects of congestion control. Initially, experiments were carried out in the *baseline* scenario; this scenario isolates the characteristics of each algorithm under the presence or absence of cross traffic, which simulates Acknowledgement (ACK) compression. None of the evaluated algorithms could fully utilize the available bandwidth, even without cross traffic, with the highest throughput measured at

approximately 74M bps on a 100M bps link. This highlights a general limitation in the ability of algorithms to maximize network utilization without additional traffic. The second scenario is called *BW scenario*. In the *BW scenario*, each analyzed algorithm is compared to itself and all others. Here, the authors evaluate intra-algorithm fairness, where flows using delay-based algorithms experience a significant reduction in throughput when sharing the bottleneck with loss-based or hybrid algorithms. The authors explain that delay-based algorithms detect congestion earlier when queues start to fill up, while loss-based algorithms keep increasing their sending rate until a loss is detected. This behavior increases the observed RTT for all flows, forcing the delay-based flow to reduce its sending rate significantly. It is reported that algorithms based on delay and BBR can maintain good levels of intra-fairness, that Cubic is more prone to oscillations, and that the convergence time of its flows is high, around the 20s, needing to be synchronized whenever losses are detected. Finally, they state that BBR, using a hybrid strategy, was not stable, contrary to what has been reported in the literature. The third scenario is called *RTT scenario with flows having different RTTs*, proposed to test the fairness of RTT between different algorithms by varying the RTTs of the flows. The authors observed that "RTT fairness" metrics are poor for all algorithms, with loss-based algorithms being those with better performance when compared to the other two groups but presenting a significant time for convergence, which is a problem for short-lived flows. The authors observed that loss-based algorithms such as CUBIC, despite claiming to be an RTT-Fairnes algorithm, also favor flows with lower RTTs. Finally, the work points out that hybrid algorithms such as BBR undergo significant dynamics in sharing between their flows, favoring those with higher RTT. This leads to complex dynamics between these flows.

In summary, with these three experiments, the authors observed that loss-based algorithms are more aggressive and induce significant packet retransmissions. In contrast, delay-based and hybrid algorithms operate without creating losses or retransmissions. Loss-based algorithms lead to higher RTTs due to queue filling, while the others utilize resources without increasing RTT. Sensitivity to ACK compression affects all algorithms, reducing throughput, particularly delay-based ones. Hybrid algorithms show variable behavior, sometimes resembling those based on loss and sometimes those based on delay. Furthermore, the study highlighted the substantial impact of algorithm selection on network latency, with loss-based algorithms contributing to increased queue lengths and, consequently, higher latency for all network flows. The study emphasizes the need for mechanisms to facilitate resource isolation between competing flows, suggesting that achieving optimal network performance goes beyond the choice of congestion control algorithm. Instead, it requires a holistic strategy that addresses the interaction between multiple algorithms and their collective impact on network resources. It is an essential step towards understanding and improving congestion control in computer networks.

Despite extensive and detailed experimentation, this paper does not address virtualized environment scenarios. Still, in our work, we use the methodology adopted by Turkovic et al. as inspiration, especially in using the "dumbbell" topology and in the detailed structuring of experiments to evaluate the performance of different congestion control algorithms. Turkovic et al.'s meticulous approach in using the categorization that divides algorithms into loss-based, delay-based, and hybrid, as well as analyzing their interactions, guided the design of our experimental framework. However, our work distinguishes itself by focusing on specific scenarios of virtualized environments, which present unique challenges and additional complexities regarding network resource management and the performance of congestion control algorithms. Furthermore, additional experiments are proposed to explore the impact of network failures and competition between multiple congestion control algorithms operating simultaneously in virtualized scenarios. By introducing new experiments, we seek to explore how virtualization impacts the effectiveness of these algorithms and their ability to deal with congestion. This guidance for virtualized environments complements the findings of Turkovic et al. and advances the discussion, highlighting the need for congestion control strategies that are optimized for such environments, considering their intrinsic characteristics and limitations.

In their work, Nguyen, Gangadhar, and Sterbenz [NGS16] adopt a methodology to evaluate the performance of various TCP congestion control algorithms within a specific context of Data Center Networks (DCNs). Using the *Network Simulator 3 (NS3)* network simulation tool, the authors constructed a fat-tree topology with carefully selected parameters to represent a typical DCN configuration. This topology includes a distribution of hosts under a network architecture that allows evaluating the performance of TCP congestion control algorithms under a variety of network conditions, including different types of traffic flows: “mice", “cat," and “elephant". Where “mice" streams are small and require quick response, such as a quick database query; “cat" are intermediate messages critical to network operation, such as state synchronization between servers; and “elephant" are large data transfers that require high bandwidth, such as a data backup. The evaluated TCP algorithms — NewReno, Vegas, HighSpeed, Scalable, Westwood+, BIC, CUBIC, and YeAH — were tested to measure four main metrics: queue length, number of dropped packets, average packet delay, and aggregate bandwidth. The results revealed significant differences in the performance of these variants, highlighting how each responds to the unique challenges presented by DCNs. For example, Vegas performed better in keeping queue length low and reducing the number of dropped packets, which are important attributes for minimizing latency and improving throughput efficiency in DCN environments. In contrast, variants such as BIC and CUBIC, known for their aggressiveness in long-distance and high latency networks, faced difficulties, resulting in a high number of dropped packets and greater queue occupancy, which is disadvantageous for the DCN scenario that demands low latency and high pre-

cision in packet delivery. The analysis suggests the need for a more adaptive algorithm design that can accommodate the dynamic characteristics and specific throughput and latency requirements of DCNs without compromising the stability and efficiency of the overall network.

This dissertation differs from the work of Nguyen et al. by adopting an experimental approach based on a physical environment rather than simulations to investigate the behavior of these algorithms, in addition to using a different topology. By utilizing physical servers with virtualized environments, this work provides a practical and applicable perspective, enabling a more in-depth and tangible analysis of the performance of TCP algorithms in real-world scenarios. This methodological difference not only enriches the understanding of the capabilities and limitations of congestion control algorithms in DCNs but also ensures that our conclusions are directly relevant to implementing and optimizing network infrastructures in data centers.

The paper of Abadleh et al. [ATB+22] also performs a comparative analysis of TCP congestion control algorithms, including the TCP Tahoe, TCP Reno, TCP New Reno, and TCP Vegas variants. Using various metrics such as throughput, latency, and packet loss rate, the authors use the *Network Simulator 2 (NS2)* simulator to create two distinct scenarios, seeking to evaluate the behavior of these algorithms under different network conditions. The research adopted a methodology that simulates TCP variants' performance under varying Constant Bit Rate (CBR) bandwidth loads, focusing on key metrics to evaluate congestion control. In the first scenario, the configuration involves a single TCP flow between two nodes, supplemented by additional CBR traffic to simulate network congestion. This setup was used to test the isolated performance of each TCP variant under increasing network load levels. The second scenario presents a more complex configuration, including two TCP flows and one User Datagram Protocol (UDP) flow, all under CBR traffic ranging from 1 Mbps to 10 Mbps. This arrangement simulates a congested network environment, allowing the assessment of competition for bandwidth between the TCP flows and the impact of UDP traffic on TCP performance. The study's results highlighted TCP Vegas as superior in the situations tested due to its ability to detect and respond to congestion early, resulting in higher throughput and lower latency in several test scenarios. Based on the findings, the authors suggest exploring machine learning techniques to identify the best congestion control method based on network traffic.

While the work provides a valuable comparison between three loss-based congestion control algorithms (TCP Tahoe, TCP Reno, and TCP New Reno) and one delay-based (TCP Vegas), it does not address hybrid algorithms that could offer additional insights into the performance in diverse network environments. Furthermore, the use of the NS2 simulator, although helpful in creating controlled scenarios, may not fully capture the complexity and variables inherent in real networks, potentially limiting the applicability of the results. Another significant limitation is the lack of consideration for virtualized

scenarios, which are increasingly common and present unique challenges due to their dynamic and shared nature, directly affecting the performance of congestion control algorithms. The limitations above are taken into account in this dissertation.

In the work of Patel et al. [PSK+20], the *NS3* network simulator is used to evaluate TCP congestion control algorithms - Newreno, Westwood, Veno, BIC, and Cubic. Using NS3 simulator, authors observed and measured the behavior of the diverse congestion control algorithms. Their analysis focused on how each algorithm adjusts congestion window size and throughput under various network conditions, essential to understanding its effectiveness in mitigating congestion. The tested algorithms include TCP variations developed to address specific limitations and improve performance in different network scenarios. For example, Newreno is known for its ability to efficiently recover from multiple packet losses in a single transmission window. At the same time, Cubic and BIC are designed for high-speed, long-haul networks. Veno, in turn, aims to optimize performance in wireless networks by identifying packet losses due to congestion or random errors. Westwood adapts the sending rate based on the estimated available bandwidth, seeking more efficient network use. The network topology used in the simulations consists of a series of TCP and UDP sources connected to receivers through three intermediate routers, configured to induce congestion conditions by limiting bandwidth and introducing delays. This configuration allowed authors to examine the algorithms' behavior in various congestion scenarios, reflecting the complexity of real networks. The results highlight significant differences in the algorithms' performance in adjusting the congestion window size and achieved throughput, with Veno demonstrating superiority in identifying and adapting to random packet losses. At the same time, BIC and Cubic proved to be more effective in networks with high bandwidth and high delay.

The study by Patel et al. presents limitations that deserve attention. Firstly, the selection of algorithms focuses on loss-based approaches, with four of the five algorithms tested following this methodology and only one (Veno) being hybrid, without including purely delay-based algorithms, like TCP Vegas. This choice limits the understanding of the performance of algorithms that use delay metrics to adjust the transmission rate, which may be more appropriate in congested network scenarios or those with high variability. Furthermore, the exclusive use of simulations through NS3, although practical and controllable, may not fully capture the complexity and nuances of real network environments, especially in virtualized scenarios, which is the focus of our work. This omission leaves a gap in understanding how congestion control algorithms behave in current widely used virtualized infrastructures, present across cloud computing platforms and data centers.

The comparison between this dissertation and existing literature is crucial for establishing its novelty and significance. Table 2.1 provides a concise comparison, highlighting differences in analysis methodologies, test environment setups, TCP congestion

control algorithms studied, and evaluation metrics.

Work	Uses Virtual-ization?	Uses Simulators or Physical Testbed?	Protocol	Categories considered	Performance Metrics
[TKU19]	No	Physical Testbed	TCP, QUIC	Delay, Loss, Hybrid	Sending Rate, Throughput, Goodput, Fairness
[NGS16]	No	Simulator (ns-3)	TCP	Delay, Loss, Hybrid	Queue length, Packet Drop Rate, Throughput, Latency
[ATB+22]	No	Simulator (ns-2)	TCP	Delay, Loss	Throughput, Latency, Packet Drop Rate
[PSK+20]	No	Simulator (ns-3)	TCP	Delay, Hybrid	Window Size, Throughput
This work	**Yes**	**Physical Testbed**	**TCP**	**Delay, Loss, Hybrid and ECN**	**Sending Rate, Throughput, Fairness, Round-Trip Time, Retransmission**

Table 2.1: Comparison of works

2.2 Discussions, Surveys, and Proposals

This section transitions the focus from practical experimentation to a theoretical one, embracing surveys, comprehensive discussions, and innovative proposals for new TCP congestion control algorithms. Such theoretical contributions enrich our understanding of the algorithms' foundational principles and inspire future research directions.

In the context of congestion control algorithms aimed specifically at DCNs, the work of C. Nandhini and Govind P. Gupta [NG23] presents an analysis of the challenges and proposed solutions in this domain. This study highlights the inadequacy of traditional TCP congestion control algorithms in dealing with the unique characteristics of traffic in DCNs, such as its bursty nature, sensitivity to delays, and throughput. Addressing specific problems such as TCP Incast, TCP Outcast, Pseudo-Congestion Effect, Buffer Pressure, and Queue Buildup, the article provides a detailed overview of the need for specialized algorithms to improve the performance of data center networks. Reviewing several categories of algorithms, including solutions based on Explicit Congestion No-

tification (ECN), non-ECN-based, routing-based, and proactive protocols, the authors provide a comprehensive overview of current strategies and their contributions to mitigating congestion in DCNs. Among the solutions examined are DCTCP, New DCTCP, D2TCP, LPD, DC-Vegas, ICTCP, TIMELY, AMTCP, CAAR, ExpressPass, ExpressPass++, MPCR, and EC4, each with its highlighted peculiarities, advantages and limitations.

This dissertation focuses on a detailed experimental analysis of the TCP Vegas, CUBIC, BBR2, and DCTCP congestion control algorithms, specifically within virtualized environments, differentiating itself from the scope of Nandhini and Gupta's article. While they provide a theoretical and comparative overview of congestion control algorithms in DCNs, this work delves into practical experimentation, examining the performance of other algorithms, defined by other criteria, under controlled conditions that simulate realistic network scenarios. This approach allows us to understand the effectiveness of these algorithms in virtualized environments and contribute specific empirical evidence to the existing literature, enriching the understanding of congestion control in highly dynamic and varied DCN contexts.

In the study "Virtualized Congestion Control (vCC)" [CRBV+16], the authors present a methodology for integrating advanced congestion control algorithms in data centers without the need to modify virtual machine (VM) operating systems or legacy applications. By implementing a translation layer in hypervisors, this method allows newly developed congestion algorithms to be applied across the board, benefiting existing applications with improvements in latency, throughput and fairness in the distribution of network resources. The central methodology of the study involves the virtualization of congestion control, in which hypervisors act as mediators between the VMs' legacy congestion algorithms and the new algorithms implemented in the hypervisor itself. This approach allows congestion control to be updated and optimized centrally without the complexity and risks associated with updating individual operating systems or applications. In the results presented, the authors demonstrate the effectiveness of vCC in improving network performance in data center scenarios. Through proofs of concept carried out on the Linux kernel and VMware's ESXi hypervisor, significant improvements were observed in fairness, performance, and the ability to control bandwidth allocations for VMs, validating the proposal that it is possible to benefit applications legacy systems with advances in congestion control algorithms without the need for direct modifications.

The study on Virtualized Congestion Control (vCC) enriches the discussion proposed in this dissertation, highlighting the relevance of the interaction between congestion algorithms and virtualized environments. By introducing an innovative methodology for implementing and evaluating congestion control algorithms without changing legacy operating systems, vCC highlights critical gaps that our research aims to investigate further. In particular, we aim to explore the performance nuances of congestion

algorithms under different virtualized network scenarios, further identifying opportunities to optimize performance in data centers.

Chi Xu et Al. [XZLC20] explores the performance of multipath congestion control in virtualized cloud environments, where data centers often implement multiple redundant paths. However, existing congestion control schemes, mostly based on single-path TCP design, cannot exploit the diversity of available paths, resulting in network underutilization and congestion on specific links. Through real-world experiments with production applications, the authors reveal that although multipath congestion control improves per-connection throughput and achieves more effective traffic balancing, it faces performance degradation when there is an abrupt increase in the number of connections or sub-flows that share paths. These challenges are attributed to virtual switches' QoS policies and interface mapping schemes. As a solution, the authors propose Virtualized Multipath Congestion Control (vMCC), which integrates ECN into virtual switches and ECN-aware multipath congestion control algorithms, demonstrating through comprehensive evaluations that vMCC improves throughput, time fairness, and RTT and energy efficiency for cloud data center traffic, benefiting typical cloud workloads. This work focuses on virtualized environments, a highly relevant scenario for our analysis. While the work highlights the importance and challenges of multipath congestion control in these environments, it also presents the predominance of single-path algorithms in current settings, which are the main focus of our analyses, showing an analysis whose goal is to broaden understanding and optimizing the performance of such algorithms in virtualized data centers. The work also highlights the importance of ECN, which is used with the DCTCP protocol in our experiments.

CHAPTER 3

Background

This chapter introduces the fundamental concepts necessary for understanding the scope of this research, including the significance of virtualized environments, the basic principles of the Transmission Control Protocol (TCP), and a review of TCP congestion control algorithms, emphasizing their application in virtualized settings.

3.1 Virtualized Environments

Virtualized environments play a critical role in the modern IT infrastructure, offering a range of benefits in terms of resource efficiency, scalability, and flexibility. They enable multiple instances of operating systems to run on a single physical platform, optimizing hardware usage and reducing operational costs. Key features of virtualized environments include:

- **Resource Optimization**: Maximizes hardware utilization by dynamically distributing resources among virtual machines.
- **Cost Reduction**: Decreases the need for hardware investment and associated maintenance costs.
- **Rapid Deployment**: Allows for quick setup and deployment of new operating system instances or applications.
- **Centralized Management**: Simplifies the administration of resources, security policies, and backups.

As illustrated in Figure 3.1, the diagram highlights the significant efficiency gains achievable through virtualization. In a traditional setup, hardware resources such as CPU cores, RAM, and HDD space are often underutilized, with low overall utilization percentages. However, when a hypervisor is employed, as shown in the virtualized environment example, these resources are dynamically allocated across multiple virtual machines, resulting in a much more optimized use of computing resources. This capability optimizes hardware usage and leads to substantial cost savings and operational efficiency, which are critical in large-scale data center operations.

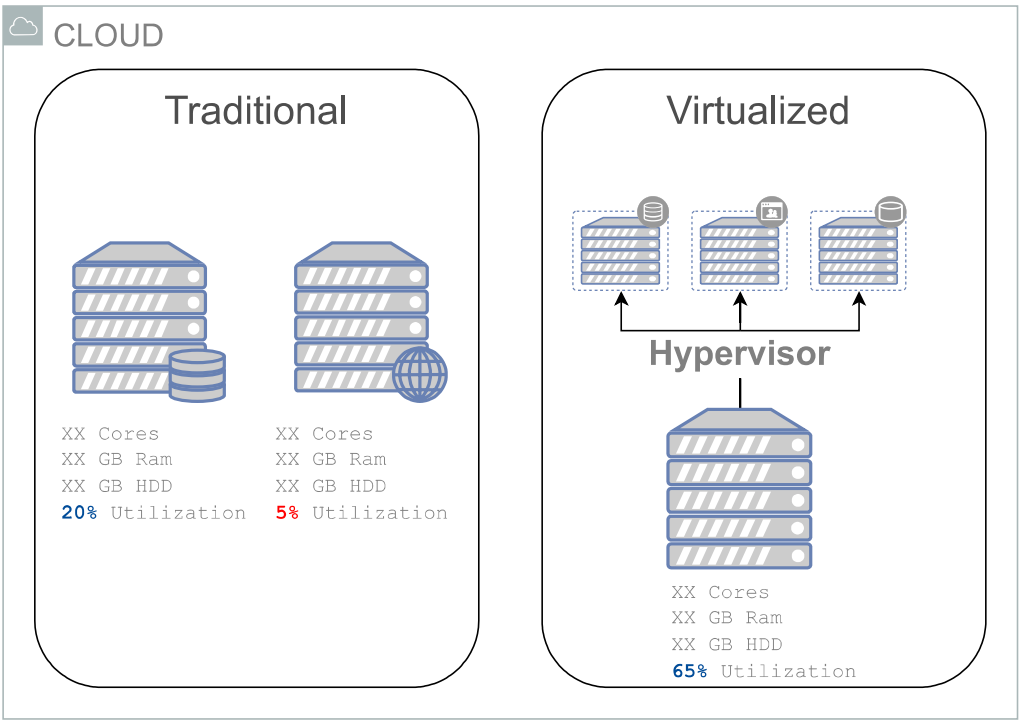


Figure 3.1: Illustration of resource utilization in traditional and virtualized environments, based on a figure by [Tec23].

This section delves into the principles underlying virtualized environments, with a focus on their application in data centers, the role of cloud computing, and the utilization of Kernel-based Virtual Machine (KVM) and Linux as core components of virtualization infrastructure.

3.1.1 Virtualized Environments in Data Centers and Cloud

Virtualization in data centers is used to optimize resource utilization, improve scalability, and increase fault tolerance. By abstracting the hardware layer (CPU, memory, storage, and network resources), virtualization allows the creation of several isolated virtual machines (VMs) on a single physical server. This approach maximizes data center efficiency by enabling a greater density of VMs per server, which reduces physical space and energy consumption. Virtualization also simplifies application management and deployment because VMs can be easily created, modified, and migrated between servers [Bel13]. Cloud computing represents the evolution of virtualized environments into scalable, on-demand computing resources delivered over the Internet. It builds

on the concept of virtualization, offering these resources as services: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [APR15]. Cloud computing abstracts the complexity of physical infrastructure, allowing users to access computing power, storage, and networking resources through a web interface without needing to manage the underlying hardware [Bad]. The scalability and flexibility of cloud services support a wide range of applications, from simple web hosting to complex, distributed applications and big data analytics.

3.1.2 Kernel-based Virtual Machine (KVM)

KVM is an open-source virtualization technology integrated into the Linux kernel. It allows Linux to function as a hypervisor, creating a virtualized environment where the kernel manages the VMs [KKL+07]. KVM converts the Linux kernel into a bare metal hypervisor that can directly access physical hardware. This integration with Linux allows KVM to take advantage of the performance and security features of the host operating system, making it a powerful and efficient solution for creating and managing VMs. KVM supports multiple guest operating systems, including Linux, Windows®, and MacOS®, and is compatible with cloud computing platforms, further highlighting its versatility in virtualized environments.

We chose a Kernel-Based Virtual Machine (KVM) for conducting experiments on congestion control algorithms in virtualized environments. KVM offers several distinct advantages that are critical to the nature and requirements of these experiments. Between them:

Enhanced Resource Management. KVM allows each virtual machine (VM) to run its own kernel [dOFFdC+22]. This architecture enables KVM to provide finer control over hardware resources through full virtualization. KVM integrates closely with the underlying hardware via the Linux kernel, allowing each VM direct access to virtualized hardware resources such as CPU, memory, and network interfaces while maintaining strong isolation between VMs.

Full Virtualization. Unlike Docker®, which uses Operating System (OS)-level virtualization to share the host's kernel with containers, KVM enables full virtualization. This means each VM operates with its own set of virtualized hardware, including a network stack that can be independently configured with different congestion control algorithms. This isolation is paramount when comparing the performance of these algorithms, as it ensures that the behavior of one VM does not inadvertently affect another, leading to more reliable and consistent experimental results [dOFFdC+22].

Suitability for Network Research. KVM's architecture, which allows for detailed configuration and monitoring of virtual network interfaces, makes it particularly suitable for network research. Researchers can implement, modify, and monitor different congestion control algorithms at a granular level within each VM, something that

container-based technologies are not designed to support as directly or efficiently.

In summary, the use of KVM in the described experiments is justified by its direct access to hardware, support for full virtualization, and enhanced performance features. These characteristics make KVM our choice for conducting detailed, controlled research on congestion control algorithms in virtualized environments, where the accuracy of performance measurement and the isolation of test conditions are paramount.

3.1.3 Linux in Virtualized Environments

Linux plays a key role in the infrastructure of virtualized environments. It is not only the underlying operating system for KVM but also the preferred guest operating system on VMs due to its open-source nature, stability, and security features. Linux's lightweight, modular architecture makes it ideal for deploying containerized applications and traditional VMs.

In summary, the theoretical basis of virtualized environments is based on the principles of abstraction, partitioning, and efficient resource management. Data centers leverage virtualization to optimize hardware usage, and cloud computing extends this concept to provide resources as scalable services. Technologies like KVM and Linux provide the tools to implement these environments effectively.

That said, it is next important to go into more detail about how packet processing works in Linux, as this can help us understand and draw better conclusions from the results obtained from the experiments.

Linux Network Path. The "Linux Network Path" refers to the path data packets take to be processed on Linux systems. These packets travel from arriving at the Network Interface Card (NIC) to the operating system, processing them, and delivering them to the receiving application. This pathway is crucial for network performance, especially in virtualization scenarios, where packet delivery efficiency directly impacts the performance of virtual machines and the services they provide.

When a packet arrives at the NIC of a Linux-based system, it is first processed by the network interface hardware, which checks for basic errors and, depending on the configuration, determines which receive queue the packet should be directed to. This is part of the multi-queue functionality of modern NICs, which allows packet processing workload to be distributed across multiple CPU cores, increasing efficiency and reducing processor bottlenecks. After being accepted by the NIC, the packet is passed to the operating system, where the Linux kernel takes control. The kernel processes packets in several steps. Firstly, it employs features like Netfilter/iptables for packet filtering, enabling actions such as dropping, modifying, or accepting packets based on security rules. Next, it determines through packet routing whether the received packet is intended for the system or should be forwarded to another network interface. Finally, for packets destined for the system, the kernel directs them to the appropriate protocol stack (e.g.,

TCP or UDP), ultimately delivering them to the user application via sockets [FRE21].

In virtualized environments, packets can have an even more complex path due to the additional abstraction layer. Before reaching the virtual machine's destination, they must pass through the hypervisor and possibly one or more virtual network bridges or switches. This additional path introduces latency and can affect throughput, especially if the hypervisor is not well-optimized or the virtual network configuration is not ideal.

Network Path Optimization in Linux systems encompasses various strategies, particularly in virtualization scenarios. Configuring hardware offloads on the NIC reduces CPU load by enabling hardware to handle tasks like TCP segmentation and flow control. Adjusting interrupt (IRQ) affinity evenly distributes packet processing across CPU cores, significantly enhancing performance. Moreover, utilizing technologies like Single Root Input/Output Virtualization (SR-IOV) in virtualized environments allows VMs direct access to NIC hardware, bypassing the hypervisor and reducing latency. Efficient packet processing in Linux is paramount for overall system performance, especially in virtualized environments, making understanding and optimizing this path essential for system administrators and network architects striving to maximize network efficiency and application performance.

3.2 Transmission Control Protocol (TCP)

Since the early days of the Internet, the Transmission Control Protocol (TCP) has played a fundamental role in communicating data between computer systems. Developed in the 1980s, it was originally specified in RFC 793 [Pos81] and emerged as part of the Transmission Control Protocol/Internet Protocol (TCP/IP) suite of protocols, which form the backbone of global network communications. TCP was developed to solve the nascent network's need for a reliable communications protocol to manage data sequence transfer between computers efficiently. As part of the Internet protocol suite, TCP operates at the transport layer of the OSI model, a crucial layer that facilitates end-to-end communication between network applications. The responsibilities of this layer include ensuring the integrity of data transmission through error checking and correction, as well as sequencing data packets to maintain the order of information flow. In essence, TCP orchestrates the complex task of dividing messages into packets, transmitting them over the network, and reassembling them at the destination, thus, ensuring that a sequence of bytes is delivered accurately and sequentially from one program to another on a different computer. Through its robust error recovery and flow control mechanisms, TCP has become instrumental in enabling the reliable and efficient exchange of information over the Internet.

How does TCP work? TCP operates following a connection-oriented model, requiring a handshake process to establish a connection before any data can be sent. This

handshake process, typically referred to as the three-way handshake, involves three steps: Synchronize (SYN), Synchronize-Acknowledge (SYN-ACK), and ACK. This process aims to ensure that both the sender and receiver are ready for data transmission and to agree on initial sequence numbers, which are used to keep track of the bytes in the stream. This mechanism establishes a reliable end-to-end communication path, preventing data transmission on an unreliable or non-existent path.

Reliable Delivery. TCP guarantees reliable data delivery through mechanisms such as sequence numbers, acknowledgments (ACKs), and retransmissions. Each byte of data sent over a TCP connection is assigned a sequence number, which the receiving end uses to reorder data segments that may arrive out of order and identify any missing segments. If a segment is not acknowledged by the receiver within a certain timeframe, the sender assumes it was lost and retransmits it. This process ensures that even if packets are lost, duplicated, or arrive out of order, the receiving application is presented with a continuous, ordered stream of data without gaps or duplicates.

Flow Control. Flow control in TCP is managed through the sliding window, which prevents a fast sender from overwhelming a slow receiver. The receiver specifies a window size in each ACK, indicating how much more data it can accept. The sender must not send data beyond what the receiver's buffer can handle, ensuring that data flow is regulated according to the receiver's capacity. This mechanism adapts to the network conditions and the receiver's processing speed, preventing buffer overflow at the receiver's end.

Full Duplex. TCP supports full duplex communication, allowing data to be transmitted in both directions simultaneously on a single connection. This is achieved by treating each direction independently, with its own sequence numbers, acknowledgments, and sliding windows. This capability allows for more efficient use of a connection, as data can flow back and forth without the need for establishing separate connections for each direction.

Connection Termination. TCP connections are terminated through a four-way handshake process. This process involves the exchange of FIN (finish) and ACK segments. Each side of the connection must separately signal that it has finished sending data and then acknowledge the other side's finish signal. This ensures that both ends have received all data before the connection is closed, preventing data loss at the end of a communication session.

Ports. TCP uses port numbers to identify sending and receiving applications. Each TCP segment header contains source and destination port numbers, which enable multiple applications to use the network simultaneously. Ports allow a single host with a single IP address to run networked services that are accessible independently, enabling multiplexing over the network and within the operating system.

Congestion Control. Finally, TCP implements congestion control algorithms to avoid saturating the network. When network congestion is detected—indicated by time-

outs or receipt of duplicate ACKs—TCP reduces its data transmission rate. Mechanisms such as a slow start, congestion avoidance, fast retransmitting, and fast recovery are used to achieve this. These mechanisms adjust the data transmission rate based on the network's capacity to handle traffic, which helps minimize packet loss and ensure efficient utilization of network resources. Congestion Control is the focus of this study, so we will go into more detail in the following sections.

Each of these features is essential to TCP's role in providing a reliable, efficient, and flexible transport layer protocol.

3.3 Congestion Control Essence

The topic of network congestion in the Transmission Control Protocol (TCP) was initiated by the work of Van Jacobson and Michael J. Karels in 1988. Their research, "Congestion Avoidance and Control," [Jac88] established the fundamental framework for congestion control mechanisms within TCP. Before this, the Internet faced substantial challenges due to its inability to manage congestion effectively, which often led to the degradation and collapse of network performance. Jacobson and Karels identified the root causes of congestion and introduced algorithms that became the cornerstone of TCP congestion control. These algorithms were based on the principle of detection and adaptation to congestion, adjusting the data transmission rate according to the conditions perceived in the network. Their work provided immediate relief from prevailing network congestion problems and set the stage for future research and development in this area.

3.3.1 Congestion

At its core, congestion in a network occurs when multiple sources send data at rates that exceed the capacity of the network to handle, leading to an overload of network resources such as routers and links. This overload causes buffers at routers to fill up and exceed their capacity, resulting in dropped packets and the need for retransmissions. The primary indicators of congestion include increased packet loss rates and growing round-trip time (RTT), which measures the time taken for a signal to be sent and the time it takes to acknowledge that signal to be received.

3.3.2 Impacts and Causes of Network Congestion

The negative impacts of network congestion can significantly degrade the quality of network services. Some of the primary adverse effects include increased latency, packet loss, and reduced throughput. The causes of network congestion are varied and often interrelated, encompassing issues such as insufficient network bandwidth relative to the

volume of transmitted data, which stands as a primary congestion aggregator. Additionally, congestion can arise from a sudden surge in data traffic, typical during peak usage times, overwhelming network resources. The scenario is further complicated by poorly optimized routing algorithms, which can lead to an uneven distribution of traffic. This results in certain paths becoming congested while others remain underutilized. Moreover, inadequate or improperly configured Quality of Service (QoS) policies contribute to the problem by failing to prioritize critical traffic, leading to network congestion that supports a mix of traffic types [TTS15].

3.4 TCP Congestion Control Algorithms

TCP congestion control algorithms play a crucial role in managing how data is sent by adjusting the transmission rate in response to the current state of the network. This dynamic adjustment prevents the network from becoming overloaded, minimizes packet loss, and ensures a fair bandwidth distribution among users.

The evolution of TCP congestion control algorithms reflects the growth and changing demands of the Internet. Initially, the Internet was a compact network serving a limited number of users, where congestion was rare. However, as the network expanded exponentially, introducing more complex and variable conditions, the original TCP algorithms had to evolve. This evolution aimed to face the challenges posed by new applications, increasing user expectations in terms of speed and reliability and the vast scale of modern networks.

From the beginnings of simple algorithms like Tahoe and Reno, designed by Van Jacobson [Jac88] to introduce primary congestion prevention and control, to sophisticated systems like CUBIC and BBR (Bottleneck Bandwidth and Round-trip propagation time) [HRX08] [CCG$^+$17], which are optimized for broadband high speed connections and minimize latency, TCP congestion control has continually adapted. These algorithms use various techniques to estimate the congestion level in the network and adjust the data transmission rate accordingly. The choice of algorithm can significantly affect the performance and reliability of connections, especially in environments with high bandwidth-delay products or where packet loss does not simply indicate congestion.

The evolution of congestion control algorithms has been significantly influenced by the transition to high-speed networks and the advent of virtualization and cloud computing. High-speed environments, especially those capable of transmitting data at a rate of 10 Gb/s, have required the development of algorithms such as HighSpeed TCP, H-TCP, and CUBIC, which are designed to utilize large bandwidths while maintaining fairness between flows efficiently. Additionally, Google's introduction of BBR marked a shift toward algorithms that base their operation on estimating network bandwidth and RTT rather than relying solely on loss signals, demonstrating significant performance

improvements in throughput and latency.

Recently, the focus has shifted to optimizing TCP congestion control for the specific characteristics of modern data centers and emerging 5G networks. This includes developing algorithms such as DCTCP, which aim to minimize queue lengths at switches within data centers, and adapting existing algorithms to operate efficiently within the constraints of 5G networks. Ongoing research and experimentation in TCP congestion control algorithms highlight the ongoing effort to adapt TCP to evolving network technologies and usage patterns, ensuring reliable and efficient data transmission over the Internet.

3.4.1 Initial algorithms

As mentioned, the basis for modern TCP congestion control was laid by Van Jacobson in 1988 [Jac88] in response to significant congestion collapse incidents that threatened the growing Internet's viability. Jacobson's work introduced several vital mechanisms that remain at the heart of TCP congestion control:

- **Slow Start:** The algorithm starts with a low data transmission rate and exponentially increases the size of the congestion window until congestion is detected, ensuring that the network is not overwhelmed by sudden spikes in data.
- **Avoid Congestion:** Once congestion is detected, this mechanism gradually adjusts the congestion window to probe network capacity, aiming to maintain the transfer rate without causing further congestion.
- **Fast Retransmission:** This enhancement allows faster recovery from packet loss without waiting for timeouts by re-transmitting lost packets after receiving a certain number of duplicate acknowledgments.
- **Fast Recovery:** A mechanism that reduces the size of the congestion window less drastically than a slow start, allowing faster recovery to optimal throughput after detecting packet loss.

These algorithms, first implemented in TCP Tahoe and later refined in TCP Reno, marked a fundamental change in network management, introducing a dynamic approach to congestion control that could adapt to changing network conditions.

3.4.2 Types of algorithms: Loss-Based, Delay-Based, and Hybrid Approaches

As TCP evolved, its congestion control strategies branched into three distinct but overlapping approaches: loss-based, delay-based, and hybrid algorithms [ATRK10]. Each

approach was developed in response to different network environments' unique challenges and characteristics. Loss-based algorithms, the first to emerge, rely on packet loss as the main indicator of congestion, decreasing data transmission rates in response to packet loss. On the other hand, delay-based algorithms, a more recent innovation, use observed increases in data packet transmission times as an early sign of potential congestion, seeking to adjust the sending rate before any loss occurs proactively. Lastly, hybrid algorithms combine packet loss signals and transmission delay to form a more comprehensive and adaptive congestion control strategy.

This section conducts a comparative analysis of these three approaches, examining the context of their emergence, the evolution of their conception, and operational relevance. The goal is to provide a nuanced understanding of how these algorithms work individually and in conjunction with each other to maintain the delicate balance of network traffic. As the Internet continues to expand and diversify, with an ever-increasing demand for bandwidth and speed, such analysis becomes essential for future innovations in TCP congestion control mechanisms.

In this work, for our experimental evaluation, we have selected representative algorithms from each category—loss-based, delay-based, and hybrid approaches—as focal points for our analysis. These chosen algorithms stand as benchmarks within their respective groups, offering a deep dive into the operational mechanics and strategic nuances that define their functionality. The selection process was guided by criteria aimed at isolating those algorithms that exemplify the core principles of their categories and exhibit a significant impact on network performance in diverse environments. This subset of algorithms, which includes examples like **CUBIC** for loss-based, TCP **Vegas** for delay-based, and **BBR2** and **DCTCP** for hybrid approaches, will be explored in greater detail compared to their counterparts. The motivation behind choosing these specific algorithms revolves around their historical significance, widespread adoption, and the innovative solutions they provide to congestion control challenges. We considered only those algorithms already implemented in the Linux kernel as selection criteria. By focusing on these algorithms, we aim to shed light on the trajectory of TCP congestion control strategies and their implications for current and future network infrastructures.

Loss-Based: Loss-based algorithms emerged since the early days of the Internet when the primary indication of network congestion was packet loss. As routers and switches reach capacity, they drop packets, a sign that the network is congested. The fundamental assumption was that packet loss directly resulted from buffer overflows in the routers, indicating that the data transmission rate exceeded the network's capacity to handle the traffic. These algorithms are particularly effective in stable wired networks, where packet loss is a reliable congestion indicator. They are simple to implement and have historically formed the backbone of TCP's congestion control mechanism. However, as networks have become faster and more complex, with variable speeds and higher bandwidth-delay products, the limitations of loss-based algorithms have become

evident. They often lead to underutilization of network capacity, especially in networks where packets can be lost due to reasons other than congestion, such as wireless networks where interference and signal attenuation can cause packet loss.

- **Tahoe**[Jac88]: This flavor was the first one to appear. Tahoe introduced essential mechanisms such as slow start and congestion avoidance. It reacts to any packet loss by starting the congestion window from scratch, using a slow start phase.
- **Reno**[Jac90]: Reno is a direct evolution of the Tahoe, improving it with the quick recovery algorithm. It differs from Tahoe in that it does not always reset the congestion window for a packet when detecting packet loss but reduces it by half, which provides a more moderate response to congestion and faster recovery.
- **NewReno**[FH99]: A further refinement of Reno, NewReno improves recovery during multiple packet losses. It waits for confirmation of all sent data before fast retransmission to exit fast recovery, which helps when more than one packet is lost in a single data window.
- **Binary Increase Congestion Control (BIC)**[XHR04]: Explicitly tailored for high-speed, long-distance networks, BIC uses a binary search approach to quickly find the window size just below that which would induce loss, allowing for rapid increases in the window size in stable conditions.
- **Cubic Congestion Control (CUBIC)**[HRX08]: An evolution of BIC, CUBIC changes the window growth function to cubic. This change allows the congestion window to grow independently of the round-trip time, **enabling more aggressive window growth after a reduction due to packet loss**, especially in networks with large bandwidth-delay products. For the category of loss-based algorithms, we chose CUBIC for our evaluation. CUBIC is currently the default algorithm in systems that use the Linux kernel, which is one of the most used algorithms. CUBIC has been in the Linux kernel since version *2.6.16*. Since kernel version *2.6.19*, CUBIC has replaced BIC-TCP as the default TCP congestion control algorithm in the Linux kernel.

  CUBIC differentiates itself from other algorithms by using a cubic window growth function to adjust the congestion window (*cwnd*) after experiencing congestion. This has two main advantages: using the available bandwidth more quickly when the current congestion window is very low and being less aggressive when the current congestion window is close to the maximum reference value ($cwnd_{\max}$). CUBIC's window growth function is a cubic function of time since the last congestion event. The cubic function is defined as [HRX08]:

$$cwnd(t) = C \cdot (t - K)^3 + cwnd_{\max} \tag{3.1}$$

where:

- $cwnd(t)$ is the congestion window size at time t after a congestion event.
- C is a constant that controls the aggressiveness of the window growth.
- t is the elapsed time since the last congestion event.
- K is the time to increase the window size to $cwnd_{\max}$, the window size before the congestion event, calculated as:

$$K = \sqrt[3]{\frac{cwnd_{\max} \cdot \beta}{C}} \tag{3.2}$$

where β is the multiplicative decrease factor in *Fast Recovery* phasis, and $cwnd_{\max}$ is the maximum window size before the congestion event. See the Figure 3.2.

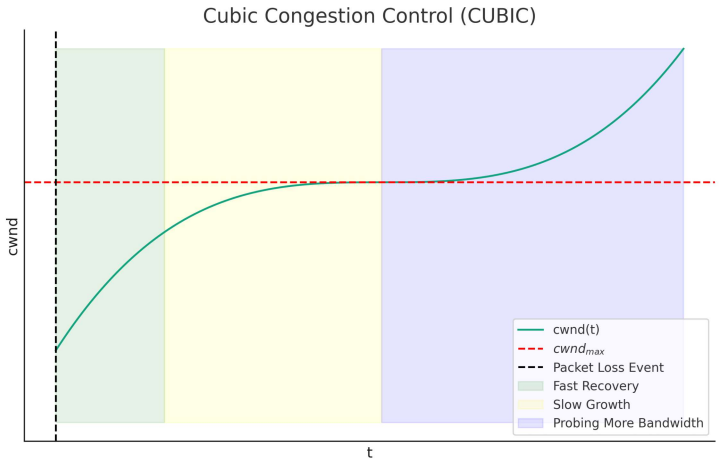


Figure 3.2: Congestion window size (cwnd) behavior using CUBIC congestion control. The regions indicate fast recovery (green), slow growth (yellow), and polling for more bandwidth (blue), with the packet loss event marked by a dotted vertical line.

CUBIC's operation can be described in several phases following a congestion event:

Unknown cwnd. Initially, if there has not yet been a packet loss event, the growth occurs following the right side of the cubic function (the blue area in Fig 3.2). This approach is more conservative than the exponential function used by other algorithms.

Congestion Detection. Like other TCP variants, CUBIC reduces its window size upon detecting packet loss (indicating congestion), typically by halving the cwnd. This is consistent with TCP's general approach to handling congestion but diverges in how it increases the cwnd after this event.

CUBIC Growth. After reducing the window size due to congestion, CUBIC enters the cubic growth phase, where it uses its cubic function to adjust the cwnd. The growth rate is independent of RTT, which makes CUBIC fairer to flow with different RTTs and more efficient on networks with high Bandwidth-Delay Products (BDPs).

Fast Convergence. To improve fairness and efficiency in bandwidth sharing, especially when competing with other flows, CUBIC implements a fast convergence mechanism. If the current $cwnd_{\max}$ is smaller than the previous $cwnd_{\max}$, indicating a recent congestion event, CUBIC reduces $cwnd_{\max}$ more aggressively to allow for faster convergence among competing flows. If it is a temporary congestion event, we will see a growth of the congestion window according to the left and right branches of the cubic function (Figure 3.2. In summary, CUBIC Fast Convergence is a mechanism designed to enhance the fairness and efficiency of bandwidth usage in TCP networks, especially where multiple data flows exist.

CUBIC offers several advantages that contribute to its effectiveness across diverse network conditions. Its scalability is highlighted by decoupling the window growth rate from the round-trip time (RTT), enabling CUBIC to perform well in high-speed and long-distance networks where traditional TCP variants might struggle. Additionally, CUBIC implements a fast convergence mechanism that promotes fairness among multiple flows competing for the same bottleneck bandwidth, ensuring a more equitable distribution of network resources. This is complemented by the cubic growth function of CUBIC, which strikes a delicate balance between aggressive window growth for rapid bandwidth utilization and cautious growth as it nears congestion, promoting stable network performance. This combination of scalability, fairness, and stability makes CUBIC a robust choice for congestion control in various networking scenarios. The main disadvantage of CUBIC is that it will never be able to use 100% of the available resources and potentially can introduce a high number of retransmissions since it relies on packet drops as an indication of congestion.

- **Hybla**: Developed to address performance degradation in TCP connections with a large RTT, such as satellite links, Hybla aims to counteract the influence of RTT, striving to give these connections a fair chance to compete with those with smaller RTTs.

- **High-Speed TCP (HS-TCP)**: This algorithm adjusts the congestion window in-

crease and adjusts parameters to be more suitable for high-speed networks with large congestion windows, allowing more precise control over the congestion window than standard TCP.

Delay-Based: Delay-based algorithms have become an alternative to loss-based algorithms, particularly to address their inefficiencies in emerging network environments. These algorithms use packet delay rather than loss to infer congestion. The rationale is that as a network approaches congestion, the queuing delay experienced by packets will increase even before any packets are dropped. By monitoring packet round-trip times (RTTs), these algorithms attempt to detect congestion early and adjust the sending rate to prevent packet loss and ensure smoother traffic flow. Delay-based algorithms are best suited for networks where low latency is crucial and packet loss does not necessarily signal congestion, such as in wireless or mobile networks. They are also relevant in high-speed networks with large bandwidth-delay products, where loss-based algorithms can significantly underperform due to their conservative nature. By using delay as an indicator, these algorithms can prevent network buffers from becoming fully occupied, thus maintaining low latency and avoiding queuing delays that can cause instability in real-time applications such as VoIP or video conferencing.

- **Fast Active Queue Management Scalable TCP (FAST))[JWL$^+$05]**: FAST maintains a small but consistent buffer occupancy at the bottleneck by using queue delay as a congestion signal. It aims for high throughput and low latency under stable network conditions.

- **Veno**[FL03]: Veno combines features from Vegas and Reno, trying to be more robust on networks with random losses, such as wireless networks, while being more aggressive on networks where packet loss does not necessarily indicate congestion.

- **Timely**[MLD$^+$15]: Designed for data center networks, Timely uses the RTT gradient, as opposed to absolute delay, to make congestion decisions, reacting to changes in RTT to manage the sending rate.

- **Vegas**[BP95]: Vegas distinguishes itself by preemptively managing congestion by monitoring RTT variations. This method allows for early congestion detection and adjustment of sending rates before packet loss becomes evident. It is more proactive than loss-based algorithms but can be less aggressive in claiming available bandwidth. For the category of delay-based algorithms, we selected TCP Vegas for our evaluation. This choice is mainly because TCP Vegas is a foundational model for numerous other delay-based and hybrid congestion control algorithms [TKU19]. Vegas has been in the Linux kernel since version *2.6.6*.

TCP Vegas uses the difference between expected and actual throughput rates to adjust its congestion window size, aiming to detect and respond to the onset of congestion before packet loss occurs. The operation of TCP Vegas can be divided into key steps [BP95]:

First, Vegas calculates the expected throughput as the size of the congestion window divided by the minimum RTT observed for the connection. The actual throughput is calculated as the size of the congestion window divided by the current RTT. By comparing these two throughput values, Vegas assesses the extent of congestion. If the actual throughput is significantly less than the expected throughput (indicating potential congestion), Vegas reduces the rate at which it increases the congestion window (cwnd). If the actual throughput is close to the expected throughput, Vegas maintains a steady increase in the congestion window, aiming for optimal utilization without contributing to congestion. If the actual throughput is greater than expected, indicating underutilization of the network, Vegas increases the congestion window to capture available bandwidth. In the congestion avoidance phase, Vegas carefully adjusts the cwnd to avoid congestion. By monitoring the RTT and keeping the congestion window increase moderate when the network begins to show signs of congestion, Vegas aims to keep queues at intermediate routers short, thus, reducing delays and avoiding packet losses.

TCP Vegas uses two critical thresholds to regulate behavior: **Alpha (α)** and **Beta (β)**. α is the lower threshold for the difference between expected and actual throughput. If the difference is less than α, Vegas concludes that there is little or no congestion and, thus, increases the congestion window more aggressively. β is the upper threshold for the difference. If the difference exceeds β, it indicates potential congestion, prompting Vegas to decrease or slow down the increase of the congestion window.

In short, TCP Vegas uses round-trip time (RTT) measurements instead of waiting for signs of packet loss to adjust its congestion window proactively. This approach enables Vegas to detect congestion sooner than loss-based mechanisms, facilitating smoother throughput and enhancing network stability by preemptively managing congestion levels. By aiming to efficiently use the available bandwidth without causing unnecessary queue buildup, Vegas not only promotes fairer bandwidth distribution among competing flows but also significantly reduces the occurrence of packet loss and retransmissions. This stability is a double-edged sword; while it contributes to a smoother network experience, it also renders the algorithm more conservative, leading to a slower growth in the congestion window (cwnd). The delay-based nature of Vegas introduces specific challenges, notably its vulnerability to pseudo-congestion effects where RTT fluctuations might

not accurately reflect actual network congestion levels. Such inaccuracies in RTT estimates can arise from factors including delayed ACKs, cross traffic, dynamic routing changes, queues within the network infrastructure, and virtualized environments, known by adding overhead due to resource sharing. Additionally, Vegas's conservative congestion window growth strategy, while reducing packet loss, can delay bandwidth utilization, particularly in mixed-traffic environments where loss-based algorithms prevail. In these scenarios, Vegas may transition to a more loss-based behavior, adapting its strategy in response to the competitive dynamics of the network. Despite these limitations, predominantly its sensitivity to RTT measurement accuracy and potential under-performance in networks dominated by aggressive loss-based algorithms, TCP Vegas offers a proactive solution for congestion control, with considerations for its operational context and the nature of competing traffic.

Hybrid: Hybrid algorithms combine the principles of loss-based and delay-based algorithms to create a more adaptive and robust approach. The emergence of hybrid algorithms has been driven by the need to address the shortcomings of loss and delay-based algorithms, especially in mixed network environments that feature a variety of technologies, speeds, and traffic patterns. The hybrid performs well in various network conditions. They are particularly relevant in heterogeneous networks, including old and new technologies, wired and wireless connections, and stable and variable links. These environments require a flexible approach to congestion control that can adapt to different congestion indicators – packet loss, delay, or a combination of both.

Hybrid algorithms take a more nuanced approach when considering multiple signals, which can lead to better decisions about when to increase or decrease the data transmission rate. This versatility makes them suitable for networks where the cause of congestion is not always clear, such as in cases where packet loss may occur due to reasons other than congestion, such as signal degradation in wireless networks, or where latency variable can be confused with network congestion, as in virtualized scenarios, where delay does not necessarily indicate congestion.

The adaptability of hybrid algorithms allows them to maintain high throughput in high-capacity networks while responding to changing network conditions, thus, ensuring efficient and fair use of network resources. For example, in networks where real-time applications coexist with bulk data transfers, hybrid algorithms can help maintain the quality of service for latency-sensitive applications while ensuring that significant data transfers are completed efficiently.

In essence, hybrid algorithms aim to capture the best of both worlds: the responsiveness of loss-based algorithms to clear indications of congestion and the pro-activeness of delay-based algorithms in anticipating and avoiding congestion. By doing so, they can optimize network performance, reduce latency, and avoid bandwidth underutilization and inefficiencies seen with purely loss- or delay-based approaches.

- **Compound TCP (CTCP)**[TSZS06]: CTCP takes a dual approach, with a delay-based component and a loss-based component working together to optimize throughput without compromising network stability. It increases the ability to utilize available bandwidth while maintaining impartiality fully.

- **Illinois**[LBS06]: This algorithm dynamically adjusts the congestion window increase and decrease factors using loss and delay indications. It offers a more responsive and less aggressive approach to congestion control, suitable for diverse network conditions.

- **Performance Oriented Congestion Control (PCC)**[DLZ+15]: PCC operates based on utility maximization. Unlike traditional TCP, which reacts to packet loss or delay events, PCC continually explores different sending rates and adopts the rate that maximizes a predefined performance objective.

- **Bottleneck Bandwidth and Round-trip propagation time (BBR)**[CCG+17]: BBR is an algorithm developed by Google® that represents a significant change in traditional TCP congestion control strategies. Unlike conventional algorithms that rely on packet loss or delay as signals to infer network congestion, BBR uses bandwidth and delay as primary metrics to control the data transmission rate. BBR has been in the Linux kernel since version *4.9*.

  BBR operates based on two fundamental metrics:

  1. **Bottleneck Bandwidth (BtlBw):** The maximum rate at which data can be transferred across the current network bottleneck without inducing queue buildup.
  2. **Round-Trip Propagation Time (RTprop):** The minimum time for a signal to travel to the destination and back represents the baseline round-trip time on the network.

  BBR works in 4 separate phases and goes through these phases to adjust the packet shipping rate:

  1. **Startup**: In this initial phase, BBR quickly probes bandwidth, exponentially increasing the sending rate with each round trip until it detects packet loss, which signals that a potential bandwidth bottleneck has been reached.
  2. **Drain**: To drain any queue that may have formed during initialization and ensure that it does not overestimate the BtlBw, BBR switches to the Drain phase, which reduces the sending rate to allow queued packets to be delivered. This helps achieve BBR's goal of minimal queue delays.

3. **ProbeBW**: BBR enters the ProbeBW phase after the queues are drained, where it periodically increases and decreases the sending rate to probe the current BtlBw. It uses a cyclical approach, spending most of its time sending data at a rate it believes matches the network bandwidth but occasionally polling above or below that rate.
4. **ProbeRTT**: Because network paths can change, BBR occasionally enters a state that reduces its sending rate to a minimum value for a brief period. This allows measuring the network's propagation delay (RTprop) in the absence of any self-induced congestion. This phase ensures the BBR's RTprop estimate is current and prevents the algorithm from persistently inflating the network queues.

BBR's decision-making is based on several algorithms that allow it to estimate BtlBw and RTprop and to adjust its pacing rate and congestion window:

- **Bandwidth Estimation**: BBR uses the delivery rate of ACKed packets to estimate the currently available bandwidth. It keeps a maximum value of recent bandwidth samples as the estimated BtlBw.
- **MinRTT Estimation**: BBR measures the RTT of each packet sent and maintains the minimum RTT seen in a time window (typically 10s) as the current RTprop estimate.
- **Pacing Rate Setting**: BBR sets the pacing rate (the rate at which packets are sent) based on the estimated BtlBw and a pacing gain factor, which is adjusted in different phases to speed up or slow down the sending to the probe's bandwidth.
- **Congestion Window Control**: Although BBR mainly controls the sending rate through pacing, it also sets a congestion window (cwnd) as a secondary measure to avoid sending too much data. The cwnd is usually set as a multiple of the estimated BtlBw and the minimum RTT, ensuring the BBR does not have more data in transit than the network can handle.

Due to its characteristics, BBR can achieve high throughput and low latency, especially in networks with high bandwidth-delay products, where traditional loss-based algorithms can perform poorly. BBR tends to reduce packet loss as it does not rely on packet loss as a signal to decrease the sending rate. It responds to actual changes in available network bandwidth more directly than loss-based algorithms.

However, in some cases, BBR can be more aggressive than loss-based algorithms, potentially leading to unfair bandwidth allocation when competing for network

capacity. It can also maintain high sending rates even under conditions of persistent congestion, potentially causing long queues on bottleneck links if network buffers are large.

- **Bottleneck Bandwidth and Round-trip propagation time version 2 (BBR2)** [CCY+19]: BBR2 is the successor to the original BBR congestion control algorithm. Although BBR was acknowledged for its ability to achieve high throughput and low latency, it had several shortcomings, especially in environments with multiple concurrent flows and handling packet loss. BBR2 was developed to address these issues and provide a fairer and more efficient approach to congestion control. For the category of hybrid-based algorithms, we selected BBR2 for our evaluation. As a relatively new addition to the congestion control landscape, BBR2 represents the cutting edge in TCP congestion control technology, with its deployment in the Linux kernel highlighting its facilitate for adoption in modern network infrastructures. With its focus on minimizing latency and improving fairness while maintaining high throughput positions, BBR2 is a compelling choice for our analysis of advanced congestion control mechanisms. BBR2 is currently being used on a small percentage of global YouTube® traffic and deployed as the default TCP congestion control for internal Google® traffic [CCY+19]. BBR2 has an implementation available for the Linux kernel, which can be installed via the official repository tutorial [1].

  BBR2 improvements. BBR2 maintains its predecessor's model-based design, focusing on bandwidth and round-trip time to make decisions, but introduces several important updates:

  - **Improved loss handling**: Traditional BBR can be too aggressive in packet loss environments, leading to unfair bandwidth allocation and possible queuing. BBR2 addresses this by incorporating loss-based signals into its model, allowing it to respond more appropriately to packet loss and thus improving its coexistence with loss-based congestion control algorithms.
  - **Ecosystem fairness**: BBR2 improves fairness by sharing bandwidth with other congestion control algorithms. It uses a more conservative approach to increase its sending rate, which helps prevent bandwidth hogging and ensures a more equitable distribution of network capacity across all streams.
  - **Enhanced startup and drain**: The startup and drain phases have been adjusted to better respond to network conditions. BBR2 can exit the Startup phase early if it detects that it has reached a bandwidth bottleneck, reducing the potential for excessive queuing. Additionally, the Drain phase has been refined to eliminate queues more effectively.

[1] https://github.com/google/bbr/blob/v2alpha/README.md

- **ProbeRTT Refinements**: BBR2 enhances the ProbeRTT phase to ensure that the minimum RTT estimate is accurate and responsive to long-term changes in the network path. This helps prevent the algorithm from persistently growing network queues.
- **Improved bandwidth estimation**: BBR2 includes updates to its bandwidth estimation algorithm to better respond to actual changes in available bandwidth, which helps it better adapt to fluctuating network conditions.
- **ECN Support**: BBR2 fully supports Explicit Congestion Notification (ECN), which allows you to react to signs of congestion before packet loss occurs. This allows maintaining high throughput while reducing the likelihood of packet loss and queuing delays. We will discuss ECN more later in this section.

BBR2 operates on a similar phase cycle to Traditional BBR, with adjustments to improve its behavior:

- Startup: Checks bandwidth and exits to Drain when it detects bandwidth saturation or packet loss.
- Drain: Reduces outstanding data on the network to the estimated BDP to ensure no queue delays are added.
- ProbeBW: Cycles through gain values to probe for bandwidth while maintaining a small amount of queuing.
- ProbeRTT: Ensures a fresh min RTT sample periodically to react to changes in the network path.

BBR2 builds on the strengths of the original BBR algorithm while addressing its weaknesses. It brings a more sophisticated approach to congestion control, considering not only the maximum achievable bandwidth but also the effects of packet loss and the need for fairness in sharing bandwidth. The improvements in BBR2 reflect an ongoing effort to optimize network throughput and latency, ensuring that the TCP congestion control algorithm remains responsive to modern networks' diverse and changing conditions.

Explicit Congestion Notification (ECN). In addition to the three main categories of TCP congestion control algorithms – loss-based, delay-based, and hybrid – there is an auxiliary mechanism known as Explicit Congestion Notification (ECN) that deserves mention. ECN extends the IP and TCP protocols, offering an alternative to traditional congestion indicators such as packet loss and delay.

ECN was developed to address the inefficiencies associated with the binary feedback loop of packet loss. It arose from the realization that preemptively signaling congestion

before packet loss occurs could avoid the abrupt reduction in throughput that typically results from loss-based congestion control mechanisms. ECN allows network nodes (such as routers and switches) to mark packets to indicate the start of congestion without dropping packets. This early warning system allows endpoints to react to congestion more quickly and smoothly without the penalties associated with packet loss, such as retransmissions and timeouts.

ECN uses the two bits of the IP header that were initially reserved but undefined in the standard protocol. The network infrastructure can set these bits to indicate that congestion is imminent. When the receiver receives a packet with ECN bits set, it notifies the sender via the TCP header's ECN-Echo flag. The sender then reacts by reducing its transmission rate, much as it would in response to packet loss, but typically with a less drastic decrease.

ECN is particularly useful on high-speed networks where recovery from packet loss can be time-consuming and detrimental to the quality of service. It is also advantageous in scenarios where packet loss is not a reliable indicator of congestion, such as wireless networks. ECN is beneficial for real-time applications sensitive to delay and jitter, as it helps avoid latency introduced by packet loss and subsequent retransmissions.

The implementation and use of ECN require that the network infrastructure and endpoints support the protocol. Although the adoption of ECN was gradual, its implementation increased as manufacturers of networking equipment and operating systems began to include ECN capabilities in their products. ECN represents a significant step towards a more agile and efficient network congestion control system, and its integration with existing TCP algorithms further increases the robustness of data communication over the Internet.

- **Data Center TCP (DCTCP)** [AGM+10]: Data Center TCP (DCTCP) is a TCP congestion control mechanism specifically designed to address the unique requirements of data center networks, where high-bandwidth, low-latency communication is critical. Developed by researchers at Microsoft®, DCTCP aims to minimize network congestion and ensure high throughput and low latency, particularly in data center environments where small, momentary congestion events can significantly impact application performance. DCTCP leverages Explicit Congestion Notification (ECN) to detect and control congestion. We also selected DCTCP for our in-depth evaluation. DCTCP was selected because it proposes to solve problems found in data center environments, in addition to being a representative of algorithms that use ECN to adjust the congestion window. Furthermore, the DCTCP is on the operating system Windows Server 2012® and has been in the Linux kernel since version *3.18*.

  DCTCP represents a hybrid approach, combining the principles of loss- and delay-based algorithms but with a unique twist provided by ECN. DCTCP operates by

leveraging Explicit Congestion Notification (ECN) within the network to detect early signs of congestion before packet loss occurs. When a switch or router in a data center network begins to experience congestion, it marks packets by setting the ECN bits in the IP header rather than dropping them. The receiver, upon detecting these ECN marks, notifies the sender by setting the ECN-Echo flag in the TCP header of its acknowledgments (ACKs). This feedback loop allows the sender to adjust its congestion window size more accurately and responsively, reducing its sending rate to alleviate congestion.

However, unlike traditional TCP algorithms that would reduce the window by half, DCTCP adjusts the window by a fraction proportional to the fraction of marked packets in a data window. This allows for much more precise sending rate control, leading to shorter queuing delays, reduced packet loss, and high burst tolerance.

DCTCP significantly reduces queue build-ups in switches and routers, thereby decreasing network latency and avoiding packet loss. This is particularly beneficial in data center environments, where even minimal packet loss and latency can degrade the performance of distributed applications. However, it also has some limitations. Implementing DCTCP requires both sender and receiver to support ECN marking, as well as switches that can be configured for ECN marking based on queue length. This can complicate deployment in heterogeneous environments. Also, DCTCP's benefits are most pronounced within the controlled environment of a data center. Outside of this context, its performance can be less predictable, especially over the Internet, where ECN marking is not universally supported. DCTCP can lead to fairness issues when coexisting with non-ECN-capable flows. Because it reduces its window size less aggressively than traditional TCP, DCTCP flows can potentially take a larger share of the bandwidth. Finally, the performance of DCTCP is highly sensitive to the configuration of the ECN marking threshold in network switches. Improper settings can lead to either excessive marking (leading to underutilization of the network) or insufficient marking (failing to prevent congestion).

DCTCP could fall into the hybrid category due to its use of ECN marks as a congestion signal, a delay-based characteristic while responding to real congestion events (albeit ECN-signaled) similar to loss-based algorithms. However, one could also argue that DCTCP belongs in its own category due to its unique response to ECN brands. It is not purely delay-based, as it does not directly measure packet transmission delays, nor is it purely loss-based, as it does not rely on packet loss as the primary signal for congestion.

Table 3.1: Detailed Summary of TCP Congestion Control Algorithms

Category	Algorithm	Key Characteristics
Loss-Based	Tahoe	Introduced essential mechanisms like slow start and congestion avoidance. Reacts to packet loss by starting the congestion window from scratch.
	Reno	Evolves from Tahoe with quick recovery, reducing the congestion window by half upon detecting packet loss, allowing faster recovery.
	NewReno	Improves on Reno by handling multiple packet losses more efficiently, waiting for all sent data confirmation before exiting fast recovery.
	BIC	Tailored for high-speed, long-distance networks, using a binary search approach to find the window size just below loss induction quickly.
	Cubic	Utilizes a cubic window growth function, enabling aggressive window growth post-loss, particularly effective in networks with large bandwidth-delay products.
Delay-Based	**Vegas**	Uses RTT variations to manage congestion proactively, increasing the window more conservatively but efficiently, aiming to prevent queue buildup.
	FAST	Maintains low latency and high throughput by keeping buffer occupancy consistent at the bottleneck, using queue delay as a congestion signal.
	Veno	Combines features from Vegas and Reno, aiming for robustness in networks with random losses and more aggressive in non-congested networks.
	Timely	Designed for data center networks, it uses RTT gradients instead of absolute delay, adjusting rates based on RTT changes.
Hybrid	CTCP)	Integrates delay and loss indications, optimizing throughput without sacrificing stability, suitable for varied network conditions.
	Illinois	Dynamically adjusts congestion window based on loss and delay signals, less aggressive and more responsive to congestion.
	PCC	Based on utility maximization, different sending rates are continually explored to maximize a performance objective independent of traditional signals.
	BBR	Uses bottleneck bandwidth and RTT as primary metrics, moving through phases to optimize bandwidth utilization and minimize queuing.
	BBR2	Enhances BBR by incorporating improvements in loss handling, bandwidth sharing, and ensuring fairness among multiple flows.
Others	**DCTCP**	Utilizes ECN to provide precise congestion feedback within data center environments, reducing latency and packet loss by adjusting the congestion window based on ECN marks.

Chapter Conclusion

This chapter explored the evolution and different approaches of TCP congestion control algorithms. They were categorized into three main groups established in the existing literature: loss-based, delay-based, and hybrid. Table 3.1 presents a detailed summary of the algorithms presented in this chapter and highlights those used in the experiments.

Loss-Based Algorithms: Historically the first ones to be implemented, these algorithms include TCP Tahoe, Reno, NewReno, BIC, and **CUBIC**. They rely on detecting packet loss as a sign of a congestion event. TCP implementations falling in this class are effective in stable networks. However, they may not perform well on networks with high latency variation (i.e., jitter) or where packet loss can occur for reasons unrelated to congestion, such as with wireless networks.

Delay-Based Algorithms: These algorithms, exemplified by **TCP Vegas**, FAST, Veno, and Timely, use delay measures such as the round-trip time (RTT) to predict and control congestion before packet loss occurs. They are especially advantageous in environments where latency is critical and packet loss is not a reliable indicator of congestion. However, they may be less aggressive in acquiring available bandwidth, which may result in underutilization in networks with mixed traffic.

Hybrid Algorithms: Combining the advantages of the first two groups, hybrid algorithms, such as TCP Compound, Illinois, PCC, BBR, and its evolution **BBRv2**, seek to adapt to various network conditions, using both loss and delay signals. These algorithms are designed to be robust in heterogeneous network environments, offering high performance, fairness, and bandwidth usage efficiency.

Auxiliary Congestion Control Mechanisms: Besides the categories above, we present the Explicit Congestion Notification (ECN) mechanism. ECN works as an extension of the IP and TCP protocols, allowing network devices to mark packets to signal congestion without dropping them, offering a smoother response to congestion. A case in point is the DCTCP algorithm, which leverages ECN in data centers for precise congestion management. This protocol combines delay and loss-based features with a unique response to ECN feedback, differentiating it from congestion control strategies.

Each class of the above algorithms makes unique contributions to network congestion management, with specific features best suited to certain types of networks and applications. These algorithms have evolved in recent years to keep up with changes in network architecture and traffic patterns, ensuring that network infrastructure can effectively support growing data demands. In summary, the selection of congestion control algorithms must consider specific network characteristics and application requirements to optimize both network performance and stability.

CHAPTER 4

Methodology

This chapter describes the adopted methodology for evaluating TCP congestion control algorithms in a virtualized data center environment. We adopt an empirical approach to analyze the performance and behavior of TCP congestion control algorithms within virtualized environments, specifically tailored to mimic the network dynamics typically found in data centers. The experimental topology is designed to simulate inter-server communication, a critical aspect of data center operations, which directly impacts the efficiency and reliability of cloud services and applications.

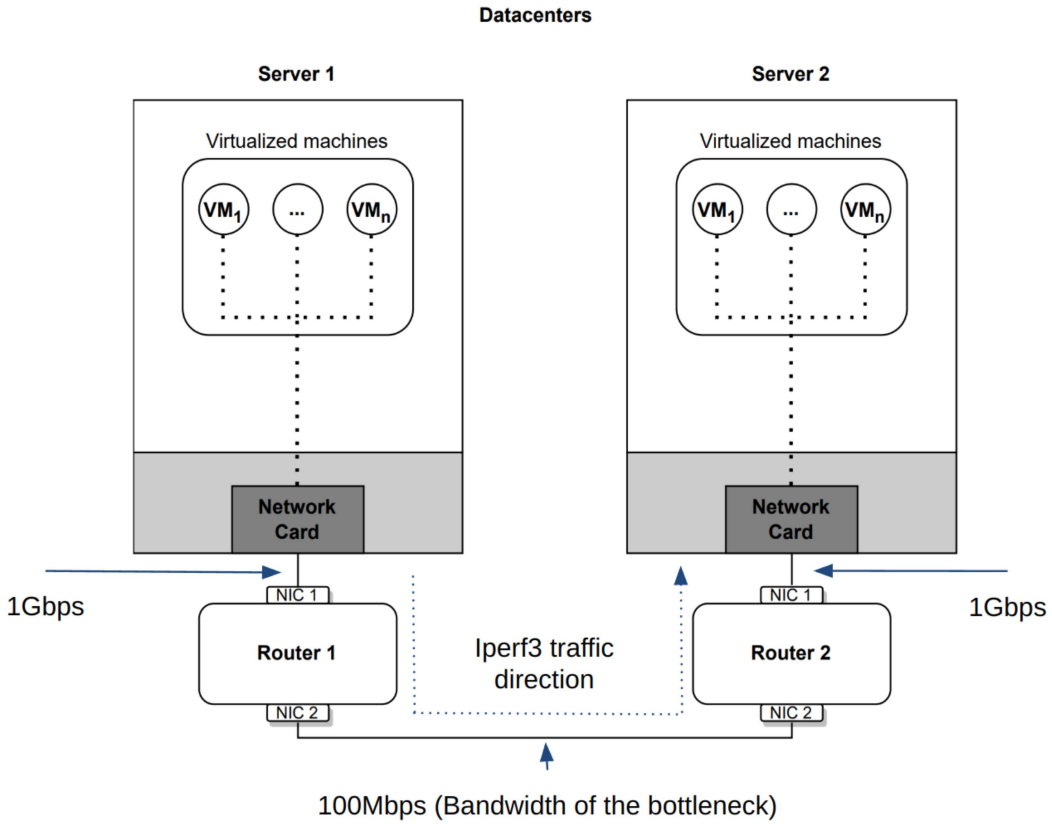


Figure 4.1: Network Topology in the Data Center Testbed.

Experimental Setup Overview. The experimental scenario comprises two primary servers, referred to as Server 1 and Server 2. Each server is connected to its respective router (Router 1 connects Server 1, whereas Router 2 connects Server 2). A direct connection was established between the two routers to facilitate inter-server communication. This setup forms a controlled network environment that allows for the precise measurement and analysis of TCP traffic between VMs hosted on the servers.

To closely emulate real-world data center conditions, each server hosts multiple KVM-type virtual machines. The KVM technology is chosen for its widespread adoption in cloud computing infrastructures and its capability to provide a scalable and secure virtualization solution. Each VM on Server 1 is configured to generate network traffic towards a corresponding VM on Server 2, creating a symmetrical flow of data that replicates the server-to-server communication patterns observed in data centers. The purpose of this setup is to provide a realistic and controlled environment for testing TCP congestion control algorithms.

Traffic Generation. The VMs on each server are configured to replicate common scenarios found in data center environments. These scenarios have been selected to reflect various operating conditions, such as varying processing loads, network latency, and data transfer rates. This emulation provides a comprehensive understanding of the performance of TCP congestion control algorithms under different, yet still realistic, data center conditions.

Evaluation. The evaluation of TCP congestion control algorithms is based on predefined metrics. These metrics reflect critical network performance aspects, including throughput, round-trip time, and packet retransmission. The algorithms are tested under various scenarios, providing insights into their effectiveness and adaptability to varying network conditions.

Physical Testbed. Unlike many studies that rely on simulated networks, this research distinguishes itself by utilizing physical testbed hardware to host virtual machines. This method ensures an enhanced level of realism in experimental outcomes by replicating real-world network environments, complete with network delays, bandwidth constraints, and hardware-specific behaviors that simulations often fail to capture accurately. Additionally, this approach allows for direct observation of network behaviors, providing deeper insights into how these factors impact performance dynamics.

Results Presentation. The results obtained from this experimental setup will be presented and analyzed systematically. This is accompanied with a detailed discussion of the performance of different TCP congestion control algorithms under various emulated scenarios, providing valuable insights into their suitability for deployment in actual data center environments.

4.1 Experiment Setup

This section investigates the specifics of our experimental infrastructure, which plays a key role in analyzing TCP congestion control algorithms in virtualized environments. The main objective of this study is to emulate a network scenario that reflects communication between servers in a data center, thus offering insights into the internal dynamics of data center networks (DCN). To this end, we meticulously designed a test environment to simulate the complexities and operational conditions of DCNs, focusing exclusively on the performance of TCP congestion control algorithms in a controlled environment that reflects real-world data center operations.

Network Infrastructure and Topology. Recall that Server 1 and Server 2 connect to two routers, Router 1 and Router 2 respectively, which in turn are connected through a dedicated link. Each server is equipped with high-performance hardware, featuring a 64-bit six-core Intel Xeon 3204 CPU, 64 Gbytes of RAM, and a 10 Gigabits per second (Gbps) network interface card (NIC). These servers are identically configured to ensure consistency in the testing environment, with their detailed specifications provided in Table 4.1. This configuration ensures ample computing and networking resources to support the demands of multiple kernel-based virtual machines (KVMs) running simultaneously on each server.

Table 4.1: Server's Hardware Specification

SO	Ubuntu 20.04.6 LTS
Kernel	5.4.0-169-generic
Processor	Intel(R) Xeon(R) Bronze 3204
RAM	64 Gbytes
Disk	ATA-DISK DELLBOSS VD SSD 240Gb
NIC	Intel x540-AT2 10Gbps

Table 4.2: Routers's Nic Specification

NIC 1	I210 1Gbps Network Connection
NIC 2	82579LM 1Gbps Network Connection (Lewisville)

The network topology, illustrated in Figure 4.1, is deliberately constructed to reflect the server-to-server connectivity prevalent in data centers. This design choice is informed by previous research in literature [DSFJ15], [dOFFdC$^+$22], [dFdC$^+$23], [TKU19], which confirms the relevance of our topology. Routers are configured with dual NICs: NIC 1 connects to the respective server and NIC 2 bridges the connection to the opposite router. The NIC's details can be found in Table 4.2. This configuration not

only replicates the interconnected nature of data center networks but also ensures that experimental conditions represent typical DCN operating environment.

Configuration and Limitations. To standardize the experimental conditions and eliminate external variables, all nodes and VMs in our setup operate using Linux as their operating system with kernel version 5.4.0-169-generic. A key parameter, the *Transmit Queue Length (txqueuelen)*, is set to 1000.

A critical aspect of our experimental design is the emphasis on the network bottleneck situated between Router 1 and Router 2 as in a Dumbbell topology. The bandwidth of this bottleneck link is artificially limited to 100 Mbps using *ethtool*, a decision that serves two purposes: creates strict conditions under which congestion control algorithms operate and ensures that any observed congestion is the result of network limitations and not VM processing capabilities. This approach allows for a focused examination of TCP congestion control algorithms, evaluating their performance in scenarios where the network, not the hardware, is the limiting factor.

A critical aspect of our methodology is the physical realization of the testbed. Unlike most literature studies that rely on emulations or simulations through tools like OMNeT++ and NS-3, our investigation employs a physical infrastructure. This approach is instrumental for several reasons. First, it allows for exploring the TCP congestion control algorithms under real-world conditions that are often difficult to replicate accurately in simulated environments. Physical testbeds offer a better level of fidelity in terms of hardware interactions, network delays, and traffic dynamics that simulations can only approximate. Secondly, by engaging with actual hardware and network configurations, our study addresses the complexities and unpredictable events inherent in real data center operations, providing insights that can both be directly applicable and highly relevant to the field.

4.2 Metrics and Tools for Experiment Evaluation

To rigorously evaluate the performance of TCP congestion control algorithms within virtualized environments, our study employs a set of carefully selected metrics and tools that can be found in Table 4.3. This selection is guided by their relevance to our research objectives and their prevalence in related literature. The algorithms listed under scrutiny—Vegas, CUBIC, BBRv2, and DCTCP—represent a diverse range of strategies for congestion control, as detailed in Section 3.4. These techniques were chosen for their distinct approaches, widespread use, and availability in the latest Linux kernel, providing a comprehensive perspective on congestion management techniques.

Table 4.3: Network Protocols and Tools

	Choice	**Reason**
Protocol	TCP	*Wide usage, built-in congestion control mechanisms, and relevance to real-world applications.*
Algorithms	VEGAS, CUBIC, BBR2, DCTCP	*These algorithms use different approaches, are widely used, and are implemented and available for use in the latest Linux kernel.*
Evaluation Metrics	Sending Rate, Throughput, Fairness, Round-Trip Time, Retransmission, Congestion Window	*Metrics used in literature works*
Tools	IPERF3, TSHARK, ETHTOOL, TC, NETEM	*Tools used in literature works*

4.2.1 Evaluation Metrics

The following metrics, commonly used in literature, serve as the foundation for our experimental analysis:

- **Sending Rate**: This metric measures the amount of data sent by a source per unit of time, typically in bits per second (bps). This metric refers to the data being sent out by the source over the network, regardless of whether it has been received or delivered at the final destination. It is crucial to evaluate how efficiently a congestion control algorithm can utilize available network bandwidth.

- **Throughput**: Throughput refers to the rate at which packets are actually successfully delivered over a network channel. It is a direct indicator of network efficiency and the effectiveness of a congestion control algorithm in managing congestion to maximize data transmission rates.

- **Throughput Fairness**: Fairness assesses how evenly network resources are distributed among multiple flows. For the evaluated algorithms, it is important to ensure that no single flow disproportionately dominates the available bandwidth, allowing for equitable resource allocation among competing flows. In this context, the Jain's Fairness Index [Jai84] was used to quantitatively evaluate the fair-

ness of resource distribution. This index is given by the formula:

$$J(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2}$$

where $x_1, x_2, \ldots, x_n$ are the throughput allocations to each flow, and n is the total number of flows. The index provides a numerical measure ranging from 0 to 1, where a value closer to 1 indicates a more equitable allocation of throughput among the flows, ensuring that no single flow is unfairly favored or disadvantaged.

- **Round-Trip Time (RTT)**: The time it takes for a packet to travel from the source to the destination and for the acknowledgment (ACK) of that packet to return to the source. RTT is a key indicator of network congestion; longer RTTs suggest increased congestion levels.

- **Retransmission**: Counts the number of packets that must be resent due to loss, excessive delay or network corruption. Frequent packet retransmissions can significantly degrade network performance, and packet drops, highlighting areas for congestion control improvement.

- **Congestion Window (Cwnd)**: The Congestion Window is a TCP state variable that limits the amount of data a source can send into the network before receiving an acknowledgment. The size of the Cwnd adjusts dynamically based on network conditions: it increases during periods of low congestion to allow more data to flow. It decreases when congestion is detected, thereby controlling the transmission rate and helping to prevent network congestion. Monitoring the Cwnd is crucial because it directly influences the sender's transmission rate and the overall throughput of the network.

These metrics are foundational for network performance evaluation, offering insights into the behavior and efficiency of congestion control mechanisms under various conditions.

4.2.2 Tools for Measurement

To gather accurate data on these metrics, we utilize a suite of established tools recognized for their reliability and precision among the network research community.

- *IPERF3*: A versatile tool for measuring maximum network bandwidth and performance. In our experiments, IPERF3 facilitates the generation of traffic patterns necessary to assess the throughput and sending rate of different congestion control algorithms.

- *TSHARK*: The command-line version of Wireshark, TSHARK, allows for the capture and analysis of network packets. It's instrumental in calculating RTT and retransmission rates by examining traffic traces before and after a bottleneck.
- *ETHTOOL*: Used for querying and controlling network driver and hardware settings. In our setup, ETHTOOL enables bandwidth limitation at the bottleneck link, a critical aspect of our experimental design.
- *Traffic Control (TC)*: A Linux utility for managing network bandwidth allocation. TC is employed to simulate network conditions such as latency and loss, essential for testing the resilience and adaptability of congestion control algorithms.
- *Network Emulator (NETEM)*: An enhancement to TC, NETEM provides advanced options for emulating network properties. It's used with TC to create a more comprehensive range of test scenarios.

Measurements are carried out using TSHARK, IPERF3, and socket statistics to capture a detailed view of network performance across different congestion control scenarios. Traffic traces are collected before and after the network bottleneck to ensure a thorough analysis. The collected data is averaged per flow, using configurable time intervals to provide a balanced and accurate representation of each algorithm's performance.

4.3 Experimental Design

The experimental design of this study is structured to systematically evaluate the performance of TCP congestion control algorithms within virtualized environments. The experimental scenarios are crafted to explore the characteristics of these algorithms under various conditions, starting from a controlled environment to more complex scenarios simulating real-world data center conditions.

As illustrated in Figure 4.2, the experimental scenarios are organized into three main categories: Baseline scenario, Basic Network Failures, and Multiple Algorithms. Each category is designed to progressively increase the complexity of the network conditions under which the TCP congestion control algorithms are evaluated. The Baseline scenario serves as the foundation, providing a controlled environment. In contrast, the Basic Network Failures and Multiple Algorithms scenarios introduce varying levels of complexity, including delayed flows and packet loss, as well as competition between different algorithms.

This section introduces and explains the experimental scenarios, focusing initially on the baseline scenario.

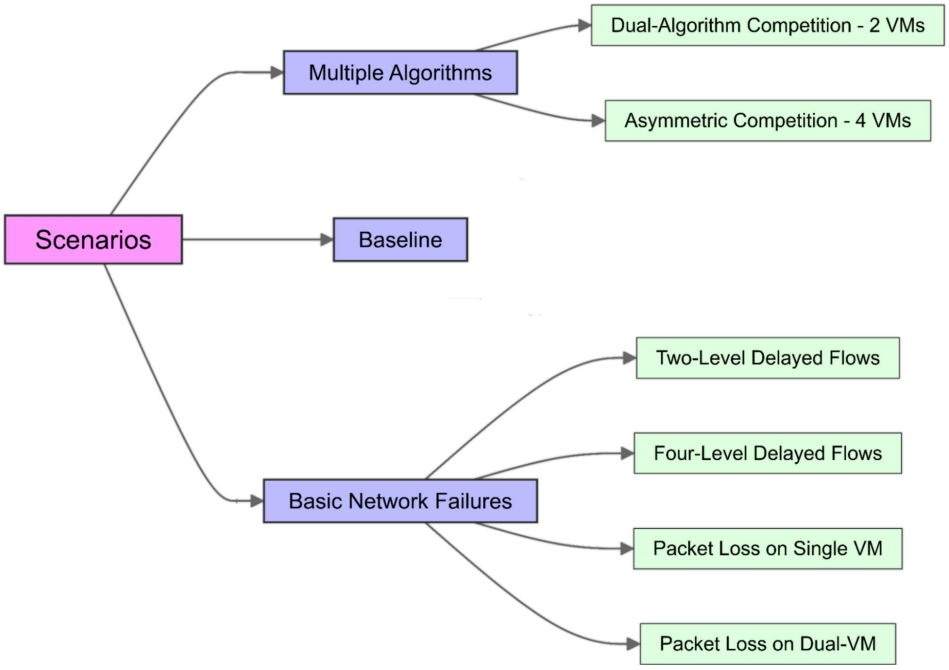


Figure 4.2: Diagram of the experimental scenarios explored in this study.

4.3.1 Baseline scenario

The primary objective of the baseline scenario is to assess the intrinsic behavior of each selected TCP congestion control algorithm—Vegas, CUBIC, BBRv2, and DCTCP—when operating independently in an environment devoid of resource competition. Each algorithm is tested separately, without competing with the others, to ensure their default characteristics are observed in isolation. This approach provides a clear benchmark for subsequent comparative analysis, where the algorithms' performances will be evaluated against each other.

Environment Setup.

- **VMs.** To assess scalability and performance under varying loads, the experiments will involve a different number of VMs on both the sender and receiver sides. The adopted configurations will include 1, 2, 4, 8, 16, and 32 VMs, allowing for a comprehensive evaluation across a spectrum of network densities.

- **Experiment Duration.** Each test within this scenario is designed to last 60 seconds. This duration is selected to ensure adequate time for the algorithms to stabilize and adapt to the network conditions and reach a steady state, facilitating reliable measurement of performance metrics.

- **Traffic Generation Command.** The `iperf3` tool is used to generate network traffic that attempts to use all available bandwidth, with the transmission rate adapting to network conditions. The command used is `iperf3 -c $VM_Addr -t 60 -i 0.1 -f m`, where:
  - `-c` specifies the IP address of the receiver VM.
  - `-t 60` sets the duration of the test to 60 seconds.
  - `-i 0.1` establishes a reporting interval of 0.1 seconds (or 100 ms), offering granular insight into the traffic behavior over time.
  - `-f m` indicates that the output format of the report should be in Megabits per second (Mbps) providing a standardized measure for analyzing throughput.

This baseline scenario is crucial for establishing a clear understanding of how each congestion control algorithm operates under ideal conditions, devoid of any external pressures such as network congestion or competition for bandwidth. It serves as a reference point for analyzing the impact of more complex network scenarios on these algorithms' performance, thereby facilitating a deeper understanding of their capabilities and limitations within virtualized environments typical of data center networks.

4.3.2 Basic Network Failures Scenario

The resilience and adaptability of TCP congestion control algorithms are critical for maintaining network performance under adverse conditions. Our second experimental scenario focuses on Basic Network Failures to evaluate these features. This scenario is meticulously designed to simulate real-world network challenges, including delays and packet losses, which are common in data centers and can significantly impact the effectiveness of congestion control mechanisms.

The main objective of this scenario is to investigate how Vegas, CUBIC, BBRv2, and DCTCP respond to controlled network failures, specifically delays and packet losses, common in real-world network environments. This evaluation helps understand each algorithm's resilience and efficiency under stress conditions.

This scenario is subdivided into four tests, each exploring different aspects of network instability.

1. **Two-Level Delayed Flow Analysis.** The objective is to compare the performance of congestion control algorithms under normal conditions flow (0 Milliseconds (ms) delay) against a flow delayed by 200 ms applied to 50% of VMs. This simulates the impact of consistent network latency on algorithm behavior
   - **VMs:** 2, 4 (equal number on sender and receiver sides).

- **Experiment Duration:** 60 seconds.
- **Traffic Generation Command:** `iperf3 -c $VM_Addr -t 60 -i 0.1 -f m`
- **Delay Implementation Command:** `tc qdisc add dev $interface root netem delay 200ms`, applied to introduce a fixed 200ms delay on the designated interface, emulating network latency.

2. **Four-Level Delayed Flow Analysis.** The objective of this setup is to assess how varying levels of delay (ranging from 100ms to 400ms) affect the performance of each algorithm. This scenario provides insights into the algorithms' adaptability to fluctuating latency.

   - **VMs:** 4 (equal number on sender and receiver sides).
   - **Experiment Duration:** 60 seconds.
   - **Traffic Generation Command:** Similar to the baseline scenario (Subsection 4.3.1) using iperf3 to generate traffic.
   - **Four-Level Delayed Flow Analysis Command:** `tc qdisc add dev $interface root netem delay $delay`, with $delay set to 100ms, 200ms, 300ms, and 400ms for different VMs, simulating variable network latency.

3. **Packet Loss Impact on Single VM.** The objective is to evaluate the algorithms' resilience to packet loss, focusing on understanding how a small percentage of packet loss (1% or 5%) impacts the network performance.

   - **VMs:** 1 (same number on sender and receiver sides).
   - **Experiment Duration:** 60 seconds.
   - **Traffic Generation Command:** Similar to the baseline scenario (Subsection 4.3.1) using iperf3 to generate traffic.
   - **Packet Loss Command:** `tc qdisc add dev $interface root netem loss $DROP_RATE`, where $DROP_RATE specifies the packet loss rate, testing the algorithms' performance under minor loss conditions.

4. **Packet Loss Impact in Dual-VM Setup.** The objective is to explore the behavior of congestion control algorithms when one VM experiences packet loss while another operates under normal conditions. This scenario helps identify how algorithms manage unequal network quality across flows.

   - **VMs:** 2 (one experiencing packet loss wheres the other VM does not).

- **Experiment Duration:** 60 seconds.
- **Traffic Generation Command:** Similar to the baseline scenario (Subsection 4.3.1) using iperf3 to generate traffic.
- **Selective Packet Loss Command:** Similar to scenario Single VM, but applied selectively to one VM to simulate differential network conditions across simultaneous flows.

Command Explanations

- **`tc qdisc add dev $interface root netem delay $DELAY`:** This command configures network emulation on the specified network interface $interface. By adding a queuing discipline (qdisc) with the netem (network emulator) module, it introduces a fixed or Four-Level Delayed Flow Analysis ($delay) to outgoing packets, enabling the simulation of network latency. We configured this on the routers to simulate specific network conditions.
- **`tc qdisc add dev $interface root netem loss $DROP_RATE`:** Similar to the delay command, this one uses the netem module to emulate packet loss on the network interface. The $DROP_RATE parameter specifies the percentage of packets to be randomly dropped, allowing for the simulation of network unreliability. We configured this on the routers to simulate specific network conditions.

The above commands, in conjunction with the traffic generated by iperf3, create a controlled environment to meticulously assess the resilience and efficiency of congestion control algorithms under various conditions of network impairments. This scenario aims to mirror the complexities of real-world networks, providing valuable insights into the algorithms' performance in practical deployments.

4.3.3 Multiple Algorithms Scenario

The focal point of this scenario is to investigate the interaction and performance dynamics when different TCP congestion control algorithms are deployed concurrently within the same network environment. This examination is crucial for understanding how these algorithms compete for network resources and manage fairness, efficiency, and stability in scenarios where multiple types of traffic coexist. It aims to reflect the complexity of real-world data center operations, where diverse applications and services may use different congestion control mechanisms.

The scenario is structured to compare the performance implications of running multiple TCP congestion control algorithms simultaneously. It does so through two specific experimental configurations:

1. **Dual-Algorithm Competition (2 VMs):** This configuration involves two virtual machines, each employing a different congestion control algorithm. This setup allows for directly comparing two algorithms under identical network conditions, focusing on their ability to manage bandwidth and maintain network performance when directly competing for resources.

2. **Asymmetric Competition (4 VMs):** This configuration expands the competitive environment by having one VM operating with a congestion control algorithm and three VMs using another congestion control algorithm. This setup tests the algorithms' performance and fairness in an uneven competitive scenario, highlighting their scalability and efficiency when the distribution of network resources is unequal among participants.

Environment Setup.

- **VMs:** The experiments are conducted with configurations of 2 and 4 VMs, corresponding to the scenarios described above. This approach allows for a scalable assessment, from minimal to more competitive network conditions.
- **Experiment Time:** 60 seconds.
- **Traffic Generation Command:** Similar to the baseline scenario (Subsection 4.3.1) using iperf3 to generate traffic.

By analyzing how different congestion control algorithms interact within the same network, this scenario sheds light on their real-world applicability in data centers where diverse traffic patterns coexist. It evaluates not just the raw performance in terms of throughput and latency but also the algorithms' ability to coexist harmoniously, maintaining fairness and efficiency without compromising the overall network stability. This analysis is pivotal for network administrators and system designers in selecting and configuring TCP congestion control algorithms that best suit their operational environments and application requirements.

4.4 Methodology Justification

This section outlines the rationale behind the developed experimental methodology, specifically the decision to run each experiment for 60 seconds with measurements taken every 0.1 seconds without conducting repeated runs.

Experimental Duration.

The duration of 60 seconds for each experiment is chosen based on the need to capture the full spectrum of the TCP congestion control algorithms' behavior within a

manageable time frame. This period allows the algorithms—Vegas, CUBIC, BBRv2, and DCTCP to exhibit their reactions to network conditions, including the initial slow start, congestion avoidance, and any necessary fast retransmit or fast recovery phases. The chosen duration strikes an fair balance between thoroughness and efficiency, ensuring that the data collected is comprehensive and practical for analysis.

$$\text{Experimental Duration} = 60 \text{ seconds} \tag{4.1}$$

Measurement Frequency. Measurements are taken every 0.1 seconds to obtain a detailed and high-resolution dataset, crucial for analyzing TCP congestion control algorithms' transient and dynamic responses to rapidly changing network conditions. At a link speed of 100 Mbps and a standard Maximum Transmission Unit (MTU) of 1500 bytes, each 0.1-second measurement period is expected to capture data from approximately 800 packets if the algorithm achieves full link utilization. This high frequency of measurement and the corresponding data volume allow for capturing the nuanced performance of the algorithms, providing insights into their efficiency, stability, and fairness under various scenarios.

$$\text{Measurement Frequency} = 0.1 \text{ seconds} \tag{4.2}$$

Lack of Repetition.

The methodology of this dissertation takes advantage of the predictability and uniform conditions of the experiment environment. Designing an experimental setup that reduces external variability to a minimum ensures the reliability of each singular execution. Given these conditions, further iterations will likely provide redundant data, thereby affirming the sufficiency of the initial experiments for drawing reliable conclusions. This strategy enhances resource and time efficiency, facilitating a wider exploration of scenarios within the project's constraints.

The experimental approach, characterized by a 60-second duration, high-resolution measurements, and a single iteration per scenario, is tailored to effectively document the dynamics of TCP congestion control algorithms in virtualized settings. This methodology balances the need for detailed, practical insights with the practical limitations of time and resource availability, ensuring the production of meaningful, applicable results. The framework of this study is consistent with precedents set by earlier research, such as that by Turkovic et al. [TKU19], Lu and Zhu [LZ15], and Abdelmoniem and Bensaou [AB21], which also adopted a similar approach to experimental design.

CHAPTER 5

Results

This chapter comprehensively analyzes the data collected from our experimental investigation into the performance of various TCP congestion control algorithms within virtualized environments, specifically under the conditions described in the experimental design scenarios. It seeks to identify, examine, and understand the impact of these algorithms on network behavior, their resource utilization efficiency, resilience in the face of network impairments, and their ability to coexist and maintain fairness when deployed concurrently. We meticulously dissect the outcomes of each experimental scenario, starting from the baseline evaluations in controlled environments to the examination of algorithmic behavior under network failures and culminating in the analysis of multiple algorithms operating simultaneously within the same network. By systematically unpacking the results, we intend to offer clear insights into the strengths and weaknesses of each congestion control algorithm.

The analysis is based on the evaluation metrics previously outlined, namely, sending rate, throughput, fairness, round-trip time, and retransmission rate. Each metric offers a unique view through which the performance and behavior of the congestion control algorithms can be assessed, providing an understanding of their operational characteristics. By detailing the outcomes of our experimental study, this chapter contributes valuable empirical evidence to the ongoing discourse on optimizing TCP congestion control algorithms for virtualized environments, particularly within the context of data center networks. The insights from this analysis were designed to inform academic research and practical applications, guiding the development of more efficient network traffic management strategies.

Each scenario analyzed in this chapter is accompanied by a table summarizing the average values for the key metrics over the entire experiment duration, which lasted 60 seconds. These averages provide a broad overview of the algorithms' performance across different conditions. While these average values offer a snapshot of the overall behavior, the variations and trends within these metrics over time are captured in the graphs presented throughout this section, offering more profound insights into how the algorithms adjust and perform dynamically during the experiments.

In calculating these averages, the entire duration of the experiment, including the initial transient or warm-up phase, was considered. This approach was chosen to maintain consistency across all experimental scenarios, ensuring that each algorithm's perfor-

mance could be directly compared under uniform conditions. By including the transient phase, the results reflect the complete lifecycle of the connection, capturing both the initial adjustment period and the stable-state performance of each algorithm.

Moreover, the transient phase was relatively short compared to the overall experiment time. This ensures that the averages still accurately represent the algorithms' performance over the total duration of the test without significantly skewing the results. Including this period provides a more comprehensive view of each algorithm's behavior, particularly in scenarios where the initial setup and adjustment periods are critical to understanding the algorithm's real-world applicability. This is especially relevant for short-lived connections, where the transient phase constitutes a significant portion of the connection's lifetime and thus has a more pronounced impact on the overall performance metrics.

5.1 Baseline Evaluation

Table 5.1: Comparative Analysis of TCP Congestion Algorithms in "Baseline" Scenario

Number of VMs	Congestion Algorithm	Average Cwnd (KBytes)	Total Retransmissions	Average RTT (ms)	Average Throughput per Flow (MB/s)	Total Data Sent (MB)	Average Sending Rate per flow (MBytes/sec)	Average Fairness Index
1	BBR2	178.11	0	12.312	11.103	677.265	11.07	-
	CUBIC	1687.07	0	136.88	11.383	694.342	11.28	-
	DCTCP	3297.38	0	140.545	11.305	689.601	11.26	-
	VEGAS	9.57	0	0.722	10.852	661.96	10.67	-
2	BBR2	169.1	0	24.975	5.706	696.127	5.7	0.965769
	CUBIC	1582.51	0	256.88	5.721	694.342	5.7	0.943689
	DCTCP	3254.04	0	261.545	5.715	689.601	5.7	0.978461
	VEGAS	10.07	0	5.722	5.767	696.127	5.7	0.994295
4	BBR2	192.21	0	55.304	2.94	720.381	2.98	0.902139
	CUBIC	1140.15	567	375.519	2.953	735.338	2.96	0.886089
	DCTCP	1218.66	539	371.071	2.945	739.098	2.98	0.879746
	VEGAS	19.11	0	6.021	2.955	721.139	2.9	0.900219
8	BBR2	187.12	739	103.613	1.569	771.741	1.59	0.826530
	CUBIC	600	831	387.702	1.541	775.101	1.57	0.752891
	DCTCP	795.57	710	402.043	1.554	784.982	1.6	0.591061
	VEGAS	26.57	0	16.913	1.556	759.553	1.53	0.806581
16	BBR2	127.55	4379	126.18	0.867	862.8	0.9	0.817516
	CUBIC	344.29	1159	409.536	0.849	866.67	0.89	0.642008
	DCTCP	336.72	2750	411.519	0.769	782.835	0.8	0.578799
	VEGAS	32.3	0	41.677	0.784	764.829	0.77	0.647575
32	BBR2	101.59	15088	211.47	0.45	902.42	0.48	0.763927
	CUBIC	208.12	2207	439.07	0.48	992.54	0.51	0.583744
	DCTCP	211.06	3686	416.31	0.47	961.52	0.50	0.469320
	VEGAS	49.02	0	113.31	0.47	945.14	0.48	0.612059

This section highlights the main results of the baseline experiments that were conducted. In this scenario, each algorithm was tested separately, without competing with the others, to ensure that their default characteristics were observed in isolation. The following discussion summarizes the results for several essential metrics: sending rate, throughput, round trip time (RTT), fairness indices for throughput, and congestion window (CWND). The summary of results for this scenario is found in Table 5.1.

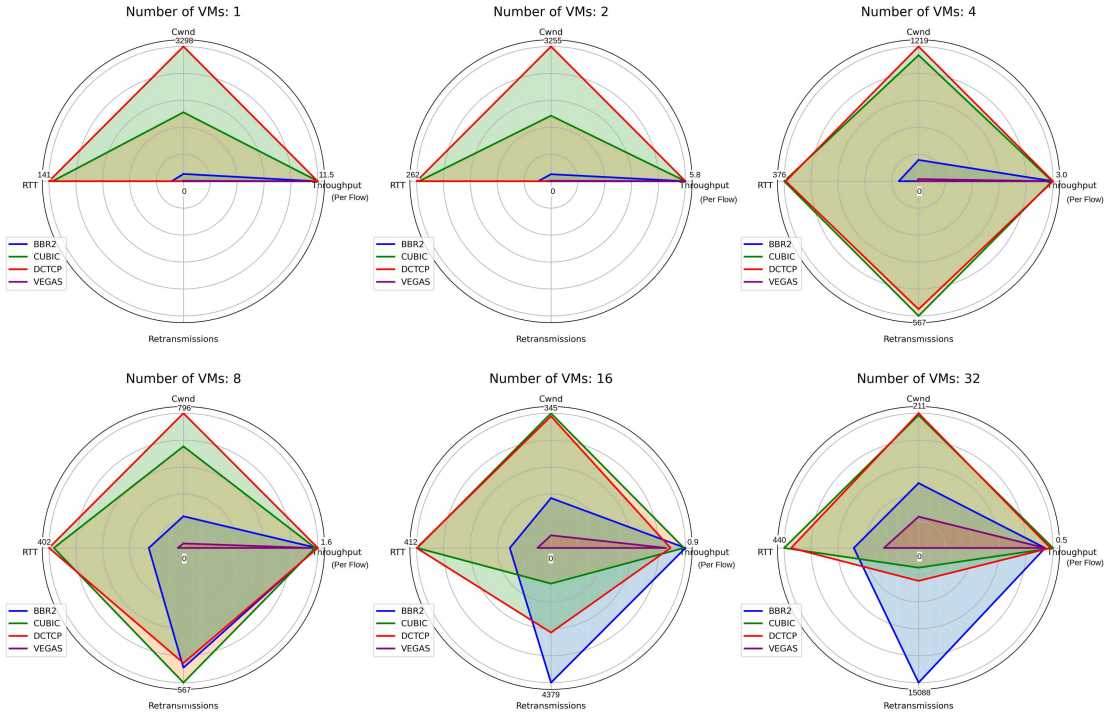


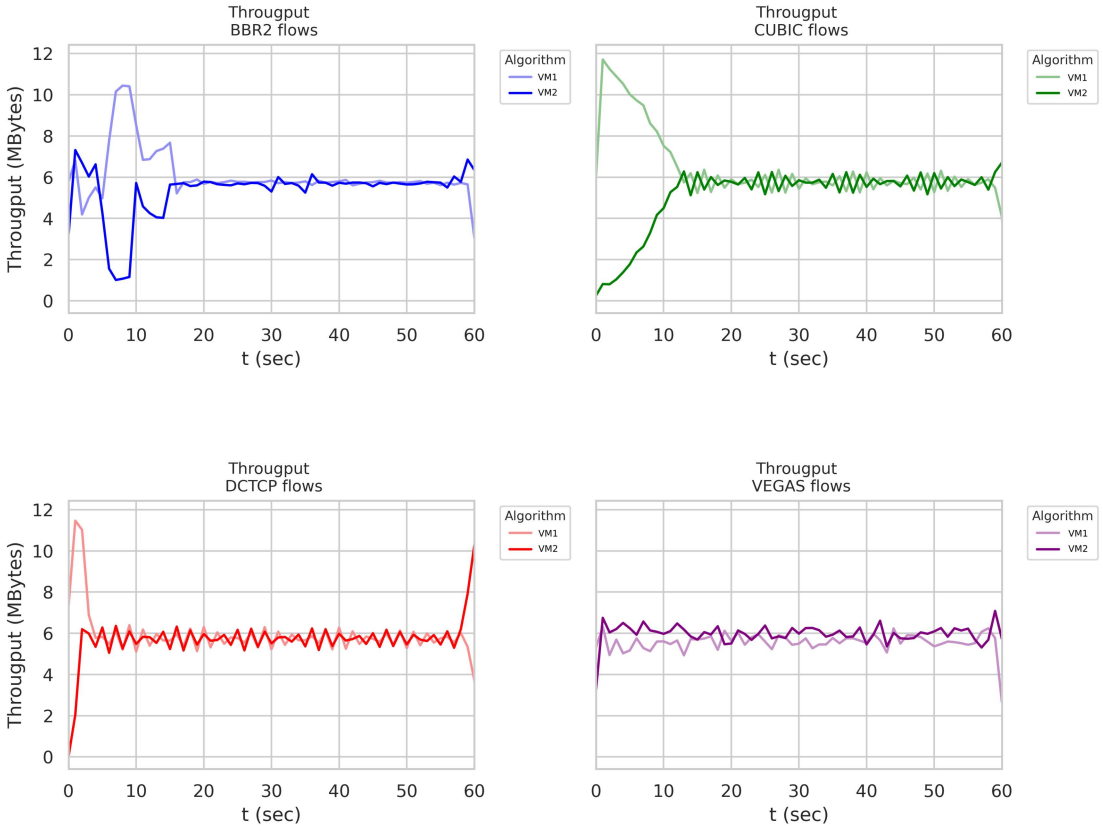
Figure 5.1: Comparison of key metrics for each experiment with a different number of VMs based on the values in Table 5.1.

1. **Sending Rate Observations**

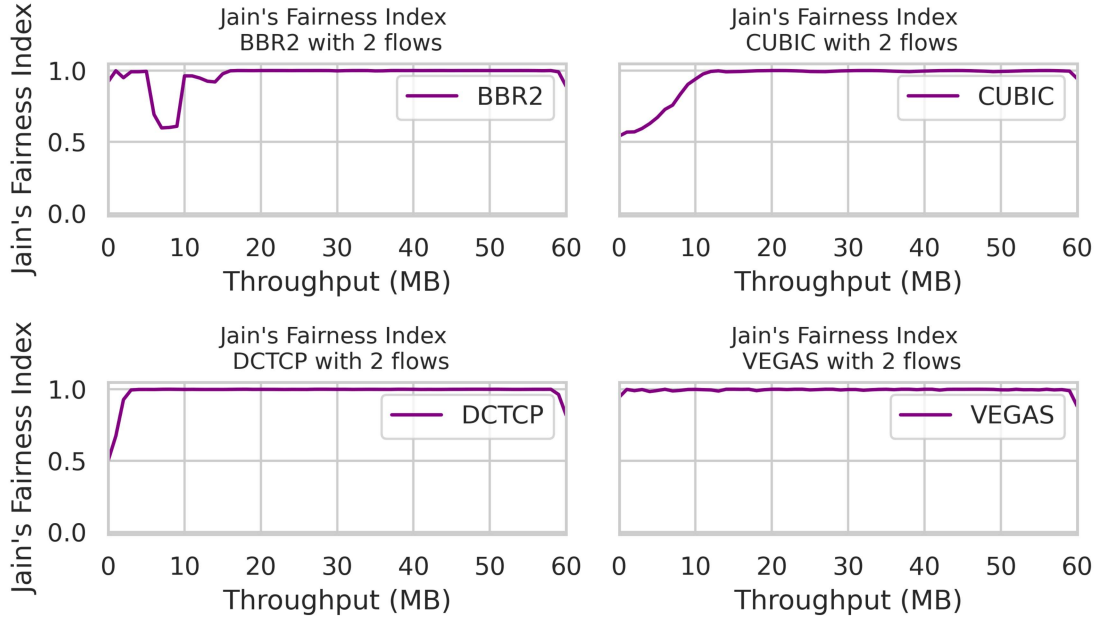
   - The sending rates for all algorithms exhibited comparable magnitudes, which indicates a general consistency in data transmission capabilities across the algorithms under test.
   - **Vegas** showed slightly lower sending rates, which aligns with its conservative design. However, this lower rate was not significantly smaller than its strong performance in other metrics.
   - **CUBIC** demonstrated higher sending rates in the configurations with 1 and 32 VMs.
   - **BBR2** and **DCTCP** displays sending rate values comparable to **CUBIC**.

2. **Throughput and Fairness**

   - With two flows, all algorithms achieved optimal fairness starting from 10 seconds into the experiments, with **Vegas** reaching this level almost imme-

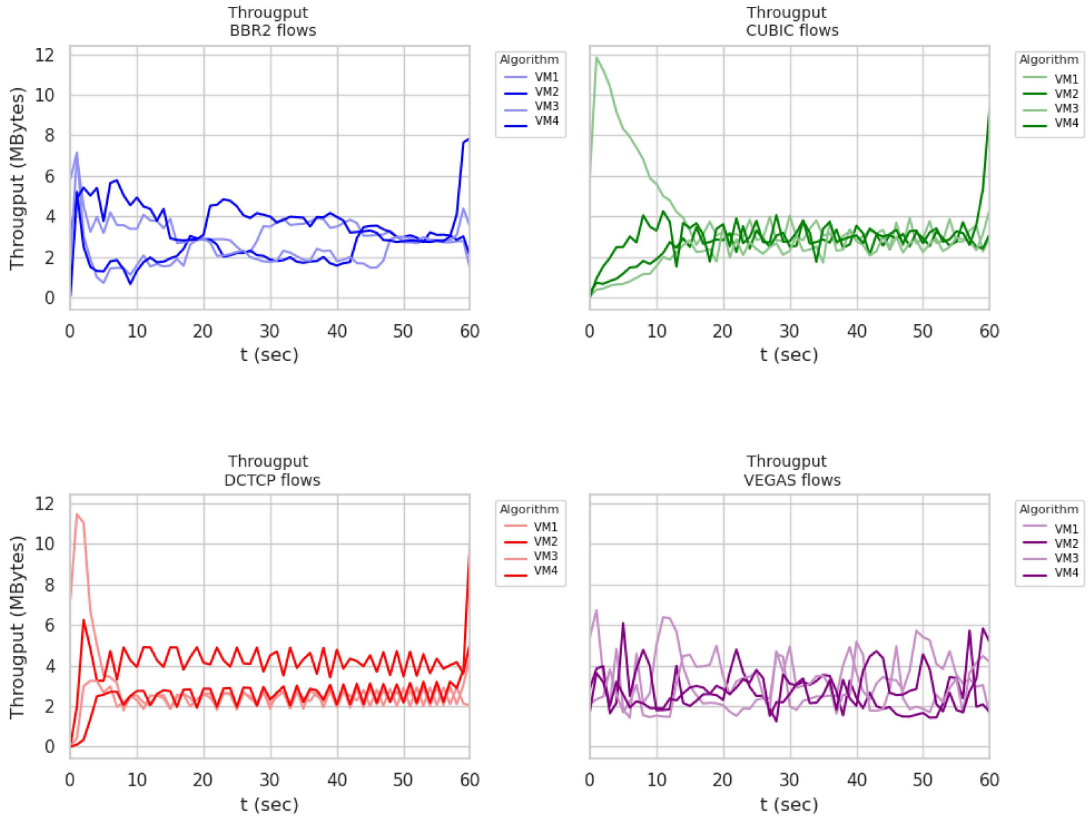


(A) Throughput for algorithms in the Baseline scenario with 2 VMs

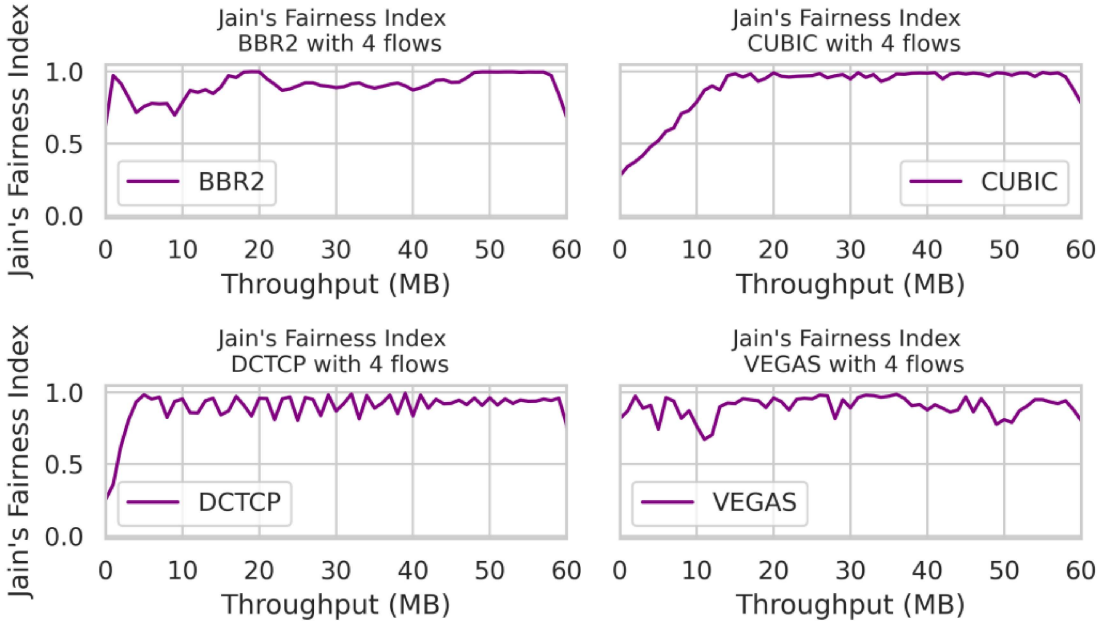


(B) Fairness for algorithms in the Baseline scenario with 2 VMs

Figure 5.2: Throughput (A) and Fairness (B) for algorithms in the Baseline scenario with 2 VMs.

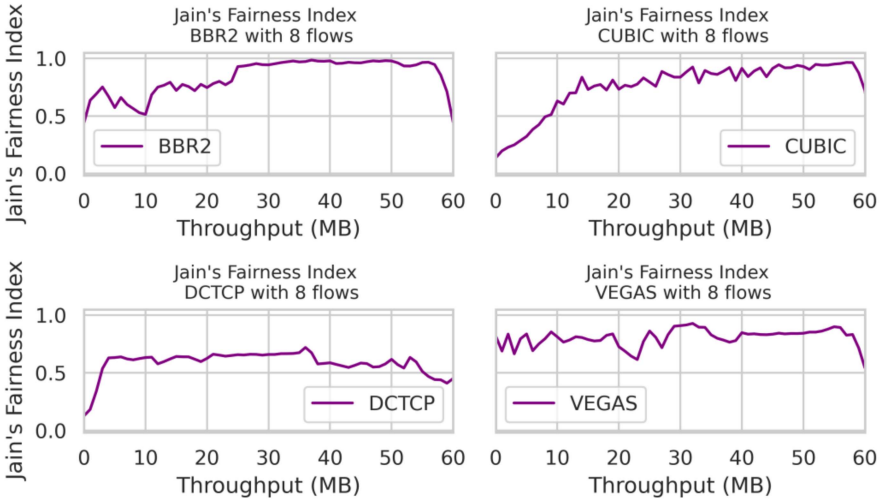


(C) Throughput for algorithms in the Baseline scenario with 4 VMs

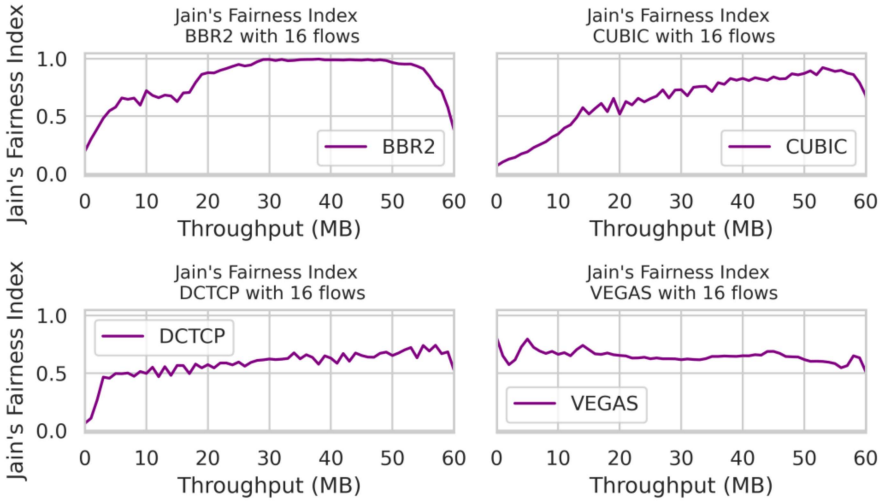


(D) Fairness for algorithms in the Baseline scenario with 4 VMs

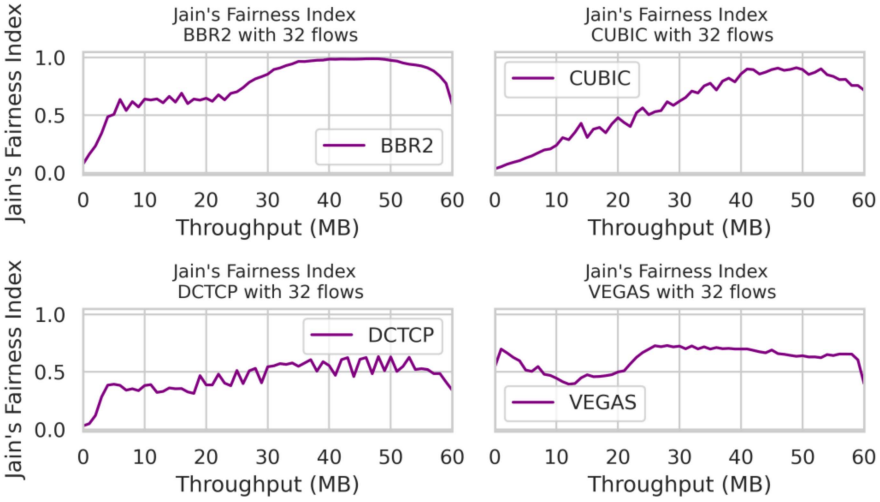
Figure 5.3: Throughput (C) and Fairness (D) for algorithms in the Baseline scenario with 4 VMs.



(A) 8 Vms



(B) 16 Vms



(C) 32 Vms

Figure 5.4: Fairness analysis for different Number of VMs: (A) 8, (B) 16, and (C) 32.

diately, which may reflect its efficiency in establishing fair bandwidth distribution early on in low-competition scenarios. This can be seen in Figures 5.2 and 5.3.

- As the number of VMs increased, **BBR2** exhibited the best fairness index, particularly when using 8 VMs onwards, suggesting its proficiency in maintaining an equitable bandwidth distribution in more competitive environments compared to others. **VEGAS** followed closely behind **BBR2** in terms of fairness, with **CUBIC** coming next as shown in Figure 5.4.

3. **Round-Trip Time (RTT)**

   - In experiments with one and two VMs, **CUBIC** and **DCTCP** recorded RTTs approximately 10x higher than **BBR2** and about 150x higher than **VEGAS**, while RTT remained close to 1ms. This vast difference points to the efficiency of **VEGAS** in maintaining low latency and the propensity of **CUBIC** and **DCTCP** to tolerate higher delays even under low competition.
   - **Vegas** maintained the lowest latency across all scenarios, with a maximum average RTT observed being 113 ms for 32 VMs, contrasting this with **CUBIC**'s highest recorded RTT of 439 ms (for 32 VMs).
   - **BBR2** exhibited an interesting pattern in the configurations of 4,8,16 and 32 VMs, initially registering high RTT values that stabilized between 20 and 30 seconds into the experiments, reflecting an adjustment period before achieving low latency comparable to **Vegas**.

   For more information on latency (RTT results), see Figure 5.6.

4. **Congestion Window (CWND)**

   - In scenarios with minimal competition, **DCTCP** and **CUBIC** used high CWND values, corresponding to their aggressive nature in claiming available bandwidth. See Figure 5.5.
   - **BBR2** and **Vegas** had considerably more conservative CWND values, aligning with their design goals and favoring a less aggressive approach to bandwidth usage. See Figure 5.5.
   - With increased competition, there was a tendency for CWND values to become more varied, which indicates adaptive responses to the changing availability of network resources.

5. **Retransmissions**

   - Retransmissions were observed starting from the experiment with four flows, with approximately 500 occurrences for both **DCTCP** and **CUBIC**. When increasing the number of flows up to 32, **BBR2** presents the highest number of absolute retransmissions; however, in Figure 5.7, we see that this higher number of retransmitted packets occurs primarily during the warm-up or transient period at the beginning of the flow.

Additional Observations

Using the values from Table 5.1, we generated the radar charts illustrated in Figure 5.1 to offer additional relevant insights. The radar graphs highlight the consistent performance of **BBR2**, which maintains a balanced profile across all metrics, except retransmissions, reflecting its robustness even with increasing competition. **CUBIC** and **DCTCP** examination exhibits a propensity for higher CWND and Throughput values at the expense of increased RTT and retransmissions, which becomes particularly higher in scenarios with higher VMs. **VEGAS**, with its conservative stance, exhibits lower transfer rates and CWND values but excels in achieving low RTT, demonstrating its efficiency in avoiding congestion without aggressive bandwidth usage.

These visual findings not only corroborate the quantitative data presented in Table 5.1 but also bring to light the different behavior of each algorithm as the network density increases.

Concluding Remarks

The baseline experiments provided a controlled platform to understand the inherent behaviors of each TCP congestion control algorithm. It was evident that more conservative algorithms like **VEGAS** and **BBR2** are designed to maintain low latency and fair bandwidth distribution, even as network demands scale. Conversely, **CUBIC** and **DCTCP** displayed a propensity for more aggressive bandwidth utilization, which may benefit throughput in scenarios where high data transfer rates are prioritized. However, they can harm fair bandwidth distribution and lead to high RTT values.

These observations lay the groundwork for further comparative analysis under more complex network conditions, which could be seen in subsequent experiment scenarios. The results from this baseline serve as a critical reference for understanding how each algorithm might perform in isolation and competition within a data center environment.

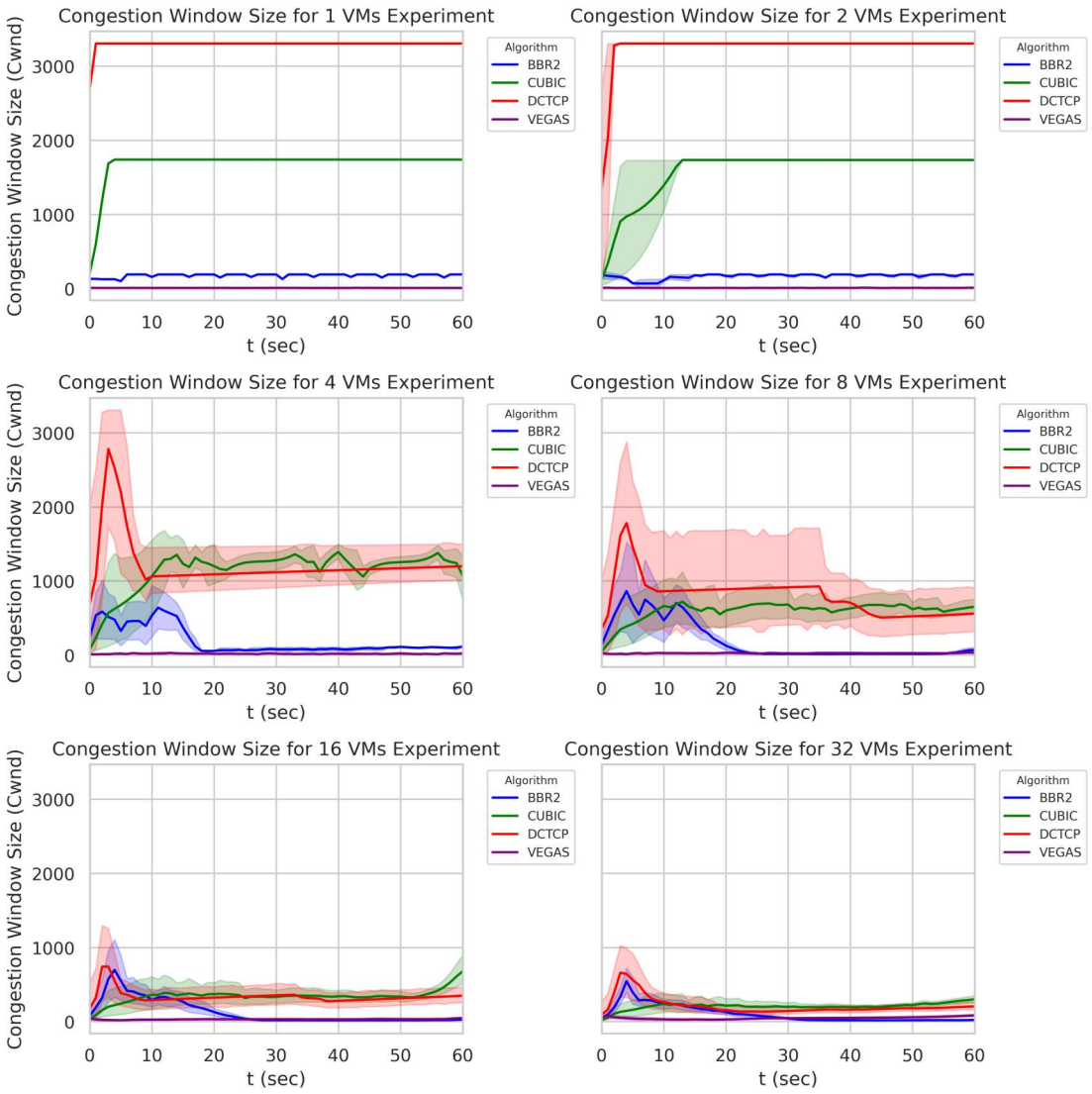


Figure 5.5: Cwnd for algorithms in Baseline scenario.

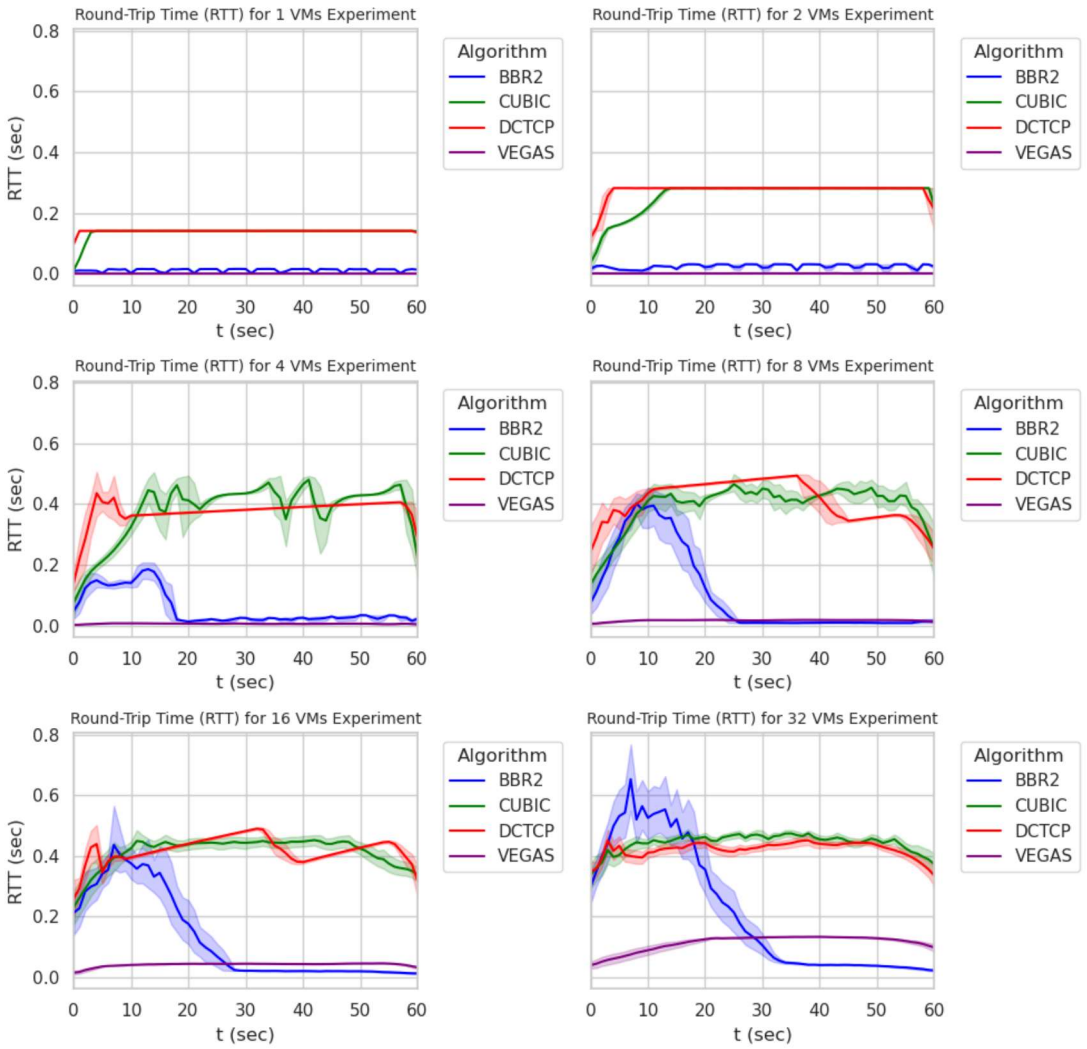


Figure 5.6: RTT for algorithms in Baseline scenario.

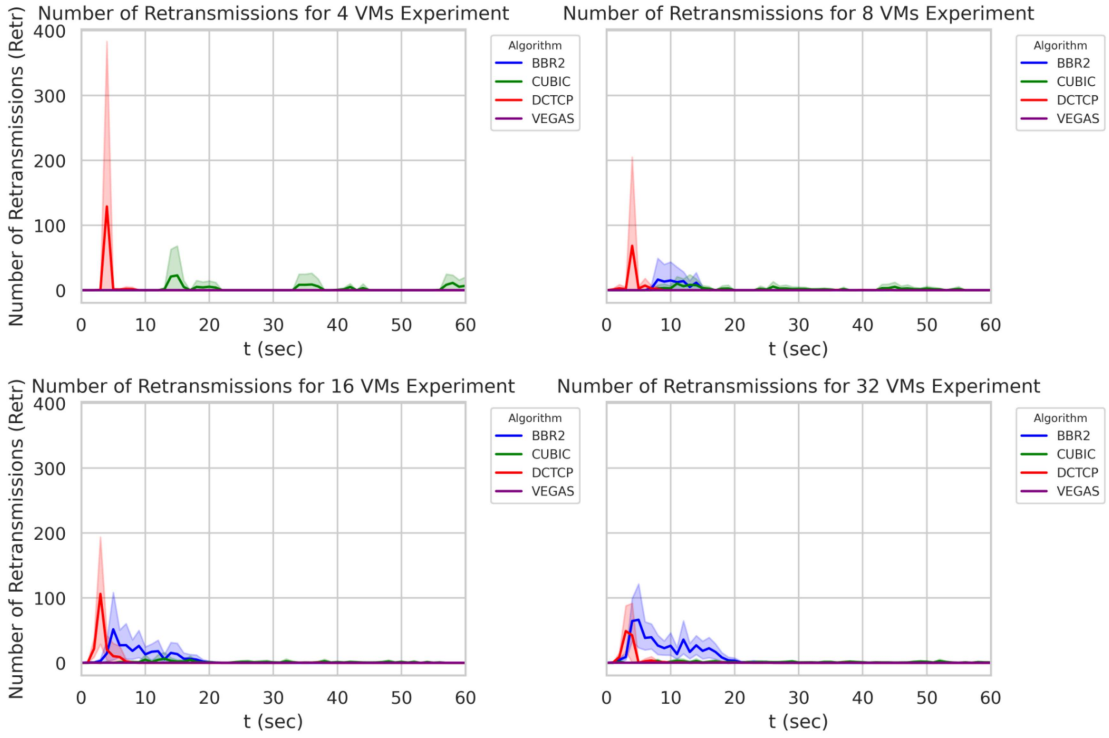


Figure 5.7: Retransmissions for algorithms in Baseline scenario.

5.2 Basic Network Failures Scenario: Two-Level Delayed Flow Analysis

Figure 5.8: Network setup for the "Two-Level Delayed Flow Analysis" scenario.

This section presents the analysis of the "Two-Level Delayed Flow Analysis" experiments. It examines the performance of the congestion control algorithms under standard conditions flow (0 ms delay) in the same environment of another flow delayed by 200 ms. The delay was applied to half of the virtual machines (VMs), simulating the impact of the competition of a flow with greater delay versus a flow with less delay on the algorithm's behavior, mainly regarding fairness. The experiments were conducted over 60 seconds with two and four flows. A summary of results for this scenario is found in Table 5.2. Figure 5.8 represents the network setup for this scenario.

1. **Sending Rate Observations**

    - With two flows across all algorithms, the one with no delay (0ms delay) achieved a higher Sending Rate. For **BBR2**, the difference between the 0ms versus 200ms flows is smaller than for the other TCP flavors, confirming its theoretical tendency of a potential to offer greater fairness. With four flows, the behavior is similar, but the difference is smaller, probably due to greater competition between the flows. See Table 5.2.

2. **Throughput and Fairness**

    - A close look at Figure 5.9 shows that for all evaluated TCP algorithms, flows with 0ms have more access to bandwidth share than flows with a delay of

Table 5.2: Comparative Analysis of TCP Congestion Algorithms in "Delayed with Two Flows" Scenario

Number of VMs	VM Number	VM Delay (ms)	Congestion Algorithm	Average Cwnd (KBytes)	Total Retransmissions	Average RTT (ms)	Average Throughput per Flow (MB)	Total Data Sent (MB)	Average Sending Rate per flow (MBytes/sec)	Average Fairness Index
2	VM1	0	BBR2	11188.59	0	19.00	8.22	501.65	8.19	0.85
	VM2	200		184122.55	0	220.00	3.31	208.40	3.36	
	VM1	0	CUBIC	102323.71	0	174.00	9.06	552.85	9.01	0.78
	VM2	200		185674.01	0	365.00	2.61	164.62	2.73	
	VM1	0	DCTCP	201140.22	0	178.00	9.05	552.18	9.02	0.78
	VM2	200		185670.98	0	364.00	2.61	164.56	2.72	
	VM1	0	VEGAS	604.13	1	1.00	9.22	562.38	9.07	0.72
	VM2	200		52269.80	190	208.00	2.03	127.73	2.06	
4	VM1	0	BBR2	3818.55	0	17.00	3.21	195.59	3.21	0.87
	VM2	200		186275.70	0	216.00	3.32	212.31	3.42	
	VM3	0		2958.11	0	17.00	2.13	129.80	2.14	
	VM4	200		187546.04	0	216.00	3.32	212.24	3.42	
	VM1	0	CUBIC	102300.58	0	344.00	5.14	323.98	5.25	0.77
	VM2	200		79029.91	42	550.00	1.53	101.03	1.64	
	VM3	0		118652.19	0	351.00	3.62	224.24	3.63	
	VM4	200		80312.90	64	594.00	1.43	88.65	1.47	
	VM1	0	DCTCP	201136.13	0	356.00	4.67	289.62	4.73	0.78
	VM2	200		175518.39	65	541.00	1.89	121.02	2.00	
	VM3	0		195864.47	0	358.00	4.28	265.30	4.33	
	VM4	200		52051.14	34	573.00	1.13	68.74	1.18	
	VM1	0	VEGAS	600.09	0	2.00	3.96	241.68	3.89	0.91
	VM2	200		55039.84	212	208.00	2.05	133.17	2.14	
	VM3	0		593.83	0	2.00	3.80	231.67	3.72	
	VM4	200		54613.95	234	210.00	2.07	132.73	2.12	

200ms. However, a different behavior is observed for BBR2 with four flows, where the flows with a delay of 200ms have access to bandwidth greater than or equal to the 0ms flows over time, unlike the experiment with the same algorithm with two flows. This behavior, where VMs with a delay of 200ms have higher throughput than those with a delay of 0ms when the number of VMs is increased from 2 to 4, can be attributed to the way BBR2 seeks to balance fairness between flows with different RTTs, to compensate for inherent delays and prevent underutilization of bandwidth by delayed streams. With only two flows, BBR2 tends to prioritize the flow with lower RTT due to faster responsiveness. However, as competing flows increase, the algorithm seeks to ensure more equitable treatment, possibly allowing flows with higher RTT to send more data. This is part of the dynamic adjustments that BBR2 makes to optimize throughput and minimize latency under various network conditions [KGCBH20].

- **BBR2** achieved the highest throughput fairness index across both delay scenarios. It was the only algorithm not requiring retransmissions, positioning it as the most reliable in the given conditions.
- **Vegas** displayed the lowest throughput fairness index with two flows but showed a substantial improvement with four flows, suggesting an enhanced ability to distribute bandwidth fairly under more competitive scenarios.
- **CUBIC** and **DCTCP** presented very similar throughput fairness indices overall and throughout the experiment duration. They performed less ef-

fectively than **BBR2** but were more equitable than **Vegas** in the two flows setup.

- We observed that the bandwidth distribution becomes more equitable across all algorithms with four flows. This is likely due to the increased competition for bandwidth. However, BBR2 remains the algorithm that achieves higher fairness values. CUBIC, which claims to achieve RTT Fairness, exhibits issues with the less equitable bandwidth distribution between flows with different RTTs. In virtualized scenarios, where there may be delays due to the inherent overhead of virtualization, this can pose a problem.

Throughput and fairness can be checked in Figures 5.9 and 5.10.

3. **Congestion Window (CWND)**

   - Analyzing the CWND, **BBR2** maintained a smaller window size for the 0ms delay flow and a larger window for the 200ms delay flow, demonstrating its adaptability to varying delay conditions. Something similar happens with **VEGAS** but on a smaller scale.
   - **Vegas** had a consistently lower CWND across both conditions (0 ms flow and 200 ms flow), aligning with its conservative congestion control strategy.
   - **CUBIC** and **DCTCP** showed higher CWND values, especially in the no-delay scenario, indicative of their aggressive stance in claiming available bandwidth.

   CWND values can be checked in Figure 5.11.

Concluding Remarks

The "Two-Level Delayed Flow Analysis" experiments within the "Basic Network Failures Scenario" provided insightful data on how each congestion control algorithm handles flows with different latency values. **BBR2** demonstrated superior performance in maintaining throughput fairness and minimizing retransmissions, suggesting its suitability for environments with varying latencies. **Vegas** showed scalability with an increased number of flows, while **CUBIC** and **DCTCP** maintained a consistent approach across different VM configurations. The results indicate that **BBR2**'s adaptive mechanisms for handling delay make it a robust choice for virtualized data center environments facing consistent latency issues, analogous to the scenario investigated here.

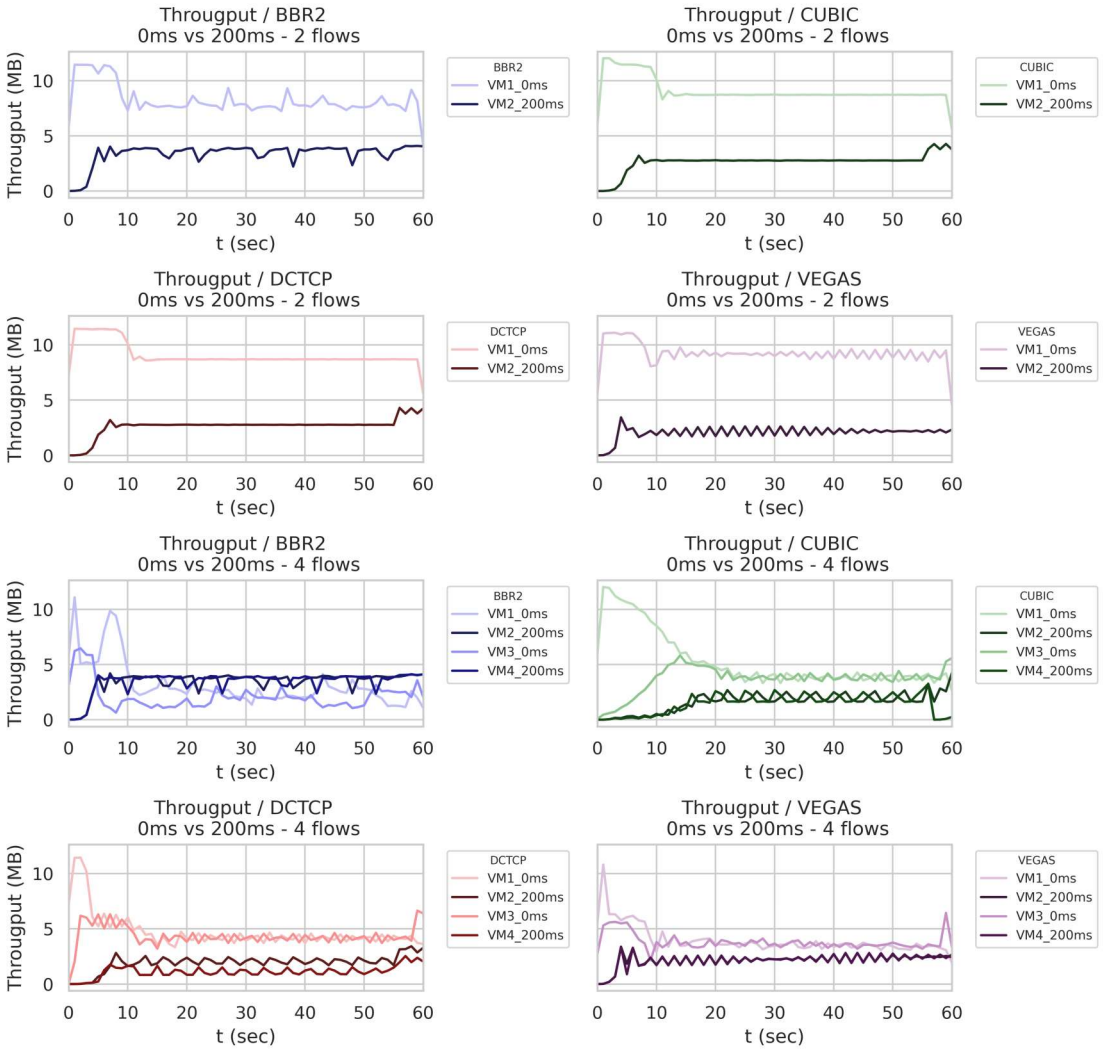


Figure 5.9: Throughput for algorithms in Delayed with Two Flows scenario.

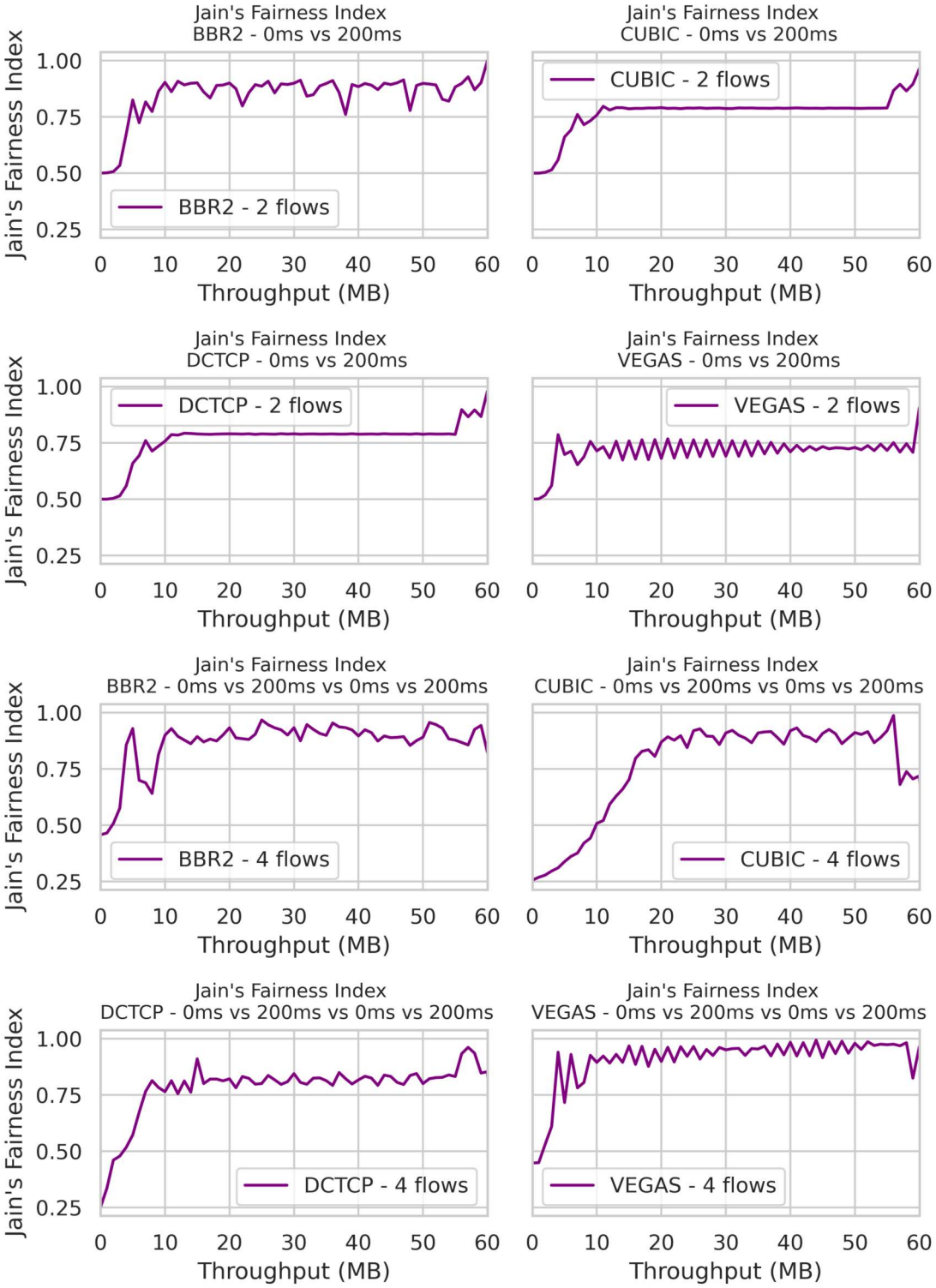


Figure 5.10: Fairness for algorithms in Delayed with Two Flows scenario.

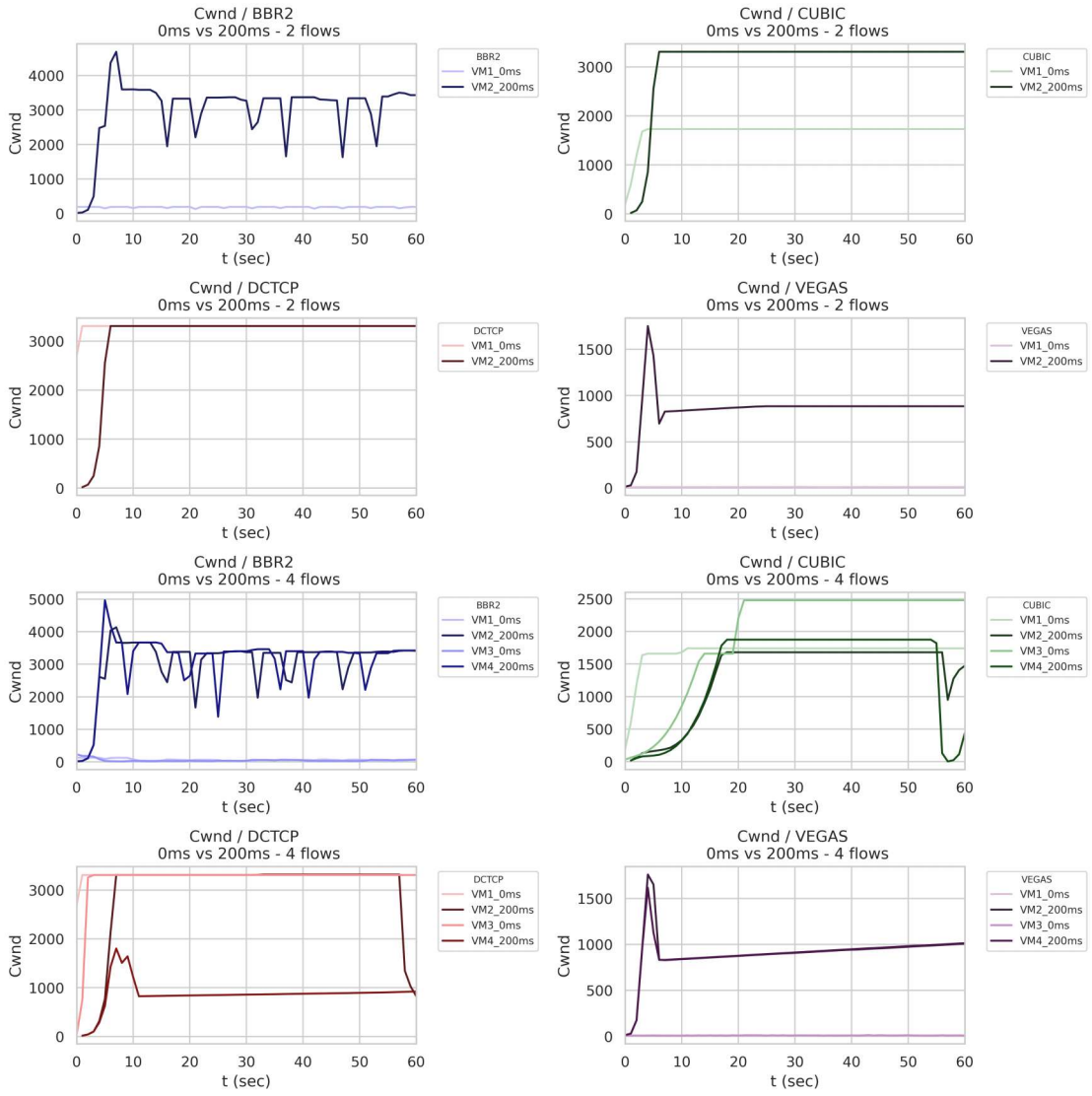


Figure 5.11: CWND (KBytes) for algorithms in Delayed with Two Flows scenario.

5.3 Basic Network Failures Scenario: Four-Level Delayed Flow Analysis

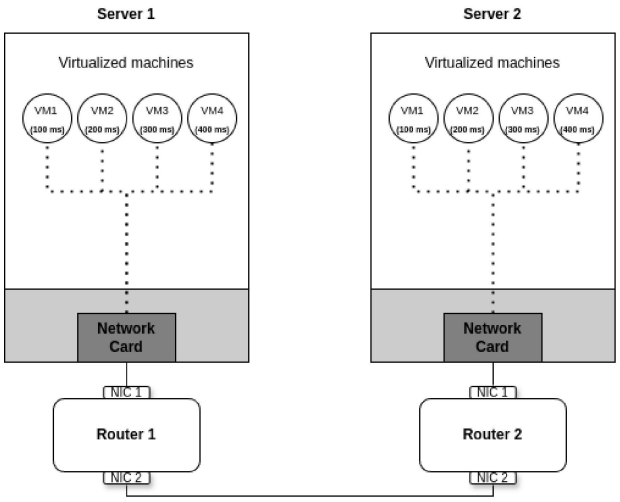


Figure 5.12: Network setup for the "Four-Level Delayed Flow Analysis" scenario.

This section presents the findings from the "Four-Level Delayed Flow Analysis" experiment, which aims to understand how the congestion control algorithms respond to fluctuating network latency. The experiment introduced a Four-Level Delayed Flow Analysis ranging from 100ms to 400ms among the VMs to simulate a dynamically latency-changing network environment. The analysis is based on the performance metrics observed over a 60-second duration. The summary of results for this scenario is found in Table 5.3. Figure 5.12 represents the network setup for this scenario.

1. **Sending Rate Observations**

    - The flow with a 100ms delay has a higher Sending Rate for all algorithms. However, for **CUBIC** and **BBR2**, the difference in the Sending rate of the 100ms flow compared to the other flows is smaller. See Table 5.3.

2. **Throughput, Fairness and Congestion Window (CWND)**

    For the "Four-Level Delayed Flow Analysis" scenario, we combine the analysis of Throughput and CWND to address our experiment's outcomes. This is due to a behavior we wish to discuss that involves both metrics. CWND dynamics can be checked in Figure 5.14. Similarly, throughput dynamics can be checked at Figure 5.13.

Table 5.3: Comparative Analysis of TCP Congestion Algorithms in “Delayed with Four Flows" Scenario

Number of VMs	VM Number	VM Delay (ms)	Congestion Algorithm	Average Cwnd (KBytes)	Total Retransmissions	Average RTT (ms)	Average Throughput per Flow (MB)	Total Data Sent (MB)	Average Sending Rate per flow (MBytes/sec)	Average Fairness Index
4	VM1	100ms	BBR2	188967.654	0	186.0	5.199	327.536	5.331	0.77
	VM2	200ms		183117.382	0	286.0	2.722	176.959	2.845	
	VM3	300ms		165500.369	0	387.0	1.877	116.392	1.929	
	VM4	400ms		153441.811	0	492.0	1.311	77.346	1.381	
	VM1	100ms	CUBIC	176615.266	0	198.0	5.234	329.717	5.374	0.76
	VM2	200ms		101669.623	208	301.0	2.722	176.918	2.831	
	VM3	300ms		89790.400	0	401.0	1.749	108.458	1.779	
	VM4	400ms		92074.295	12	504.0	1.321	77.934	1.383	
	VM1	100ms	DCTCP	194601.361	0	131.0	6.526	411.142	6.707	0.59
	VM2	200ms		54502.489	252	236.0	1.900	123.509	1.988	
	VM3	300ms		52835.249	89	347.0	1.294	80.210	1.333	
	VM4	400ms		51976.732	144	470.0	0.949	55.962	1.004	
	VM1	100ms	VEGAS	389651.847	0	146.0	6.118	385.455	6.289	0.65
	VM2	200ms		47457.408	156	248.0	1.601	104.087	1.686	
	VM3	300ms		93444.234	31	353.0	2.215	137.342	2.258	
	VM4	400ms		47943.467	29	456.0	0.882	52.043	0.935	

- All algorithms exhibit discrepancies in bandwidth distribution, favoring those with lower RTT. However, within this context, we have **BBR2** and **CUBIC** emerged as the best-performing algorithms in maintaining throughput fairness across delayed flows. **CUBIC** took about 20 seconds to adjust the flow bandwidth, indicating an initial adjustment period before stabilizing. See Figure 5.13.
- Due to its characteristic of being a delay-based algorithm, **Vegas** tends to “misinterpret” the delay as congestion, an expected outcome for this algorithm. Nevertheless, its fairness indices throughout the experiment remain marginally higher than those of **DCTCP**. See Figure 5.13.
- **DCTCP**, **VEGAS**, and **CUBIC** exhibited higher CWND for the 100 ms delay flow compared to other flows. Among these, **CUBIC** showed the smallest difference in CWND sizes between the different delays, while **VEGAS** showed the largest discrepancy.
- We can see that **BBR2** demonstrates an interesting behavior. The charts in Figure 5.13 show that throughput inversely correlates with the delay introduced; flows with lower delays achieve higher throughput initially. However, after 10s, the throughput converges to less distant values, leading to a more equitable bandwidth distribution across all flows.

With regard to BBR2, we observed a tendency for CWND size to fluctuate (Figure 5.14). On average, these CWND values remain remarkably close, with the most substantial difference and variation observed between the 100ms and 400ms streams reaching only around 20% (check the table 5.3).

This behavior can be explained by the fact that BBR2 mainly estimates the avail-

able network bandwidth (BtlBw) to calculate the congestion window (CWND) but also considers the minimum round-trip delay (RTprop) when adjusting CWND. By dynamically optimizing network resource allocation based on real-time bandwidth assessments rather than using delay metrics alone, BBR2 effectively aligns with its goal of operating in the Kleinrock's optimal point [CCG+17, CCY+19]. This strategy helps prevent excessive queue buildup and aims to optimize performance.

In virtualization environments, where VMs may experience different latency due to hypervisor actions or resource contention, **BBR2** responsively adapts to RTT variations. This adaptability proves crucial in scenarios involving four streams, as **BBR2**'s ability to adjust to the competitive dynamics between VMs is essential. The ProbeRTT phase, during which **BBR2** temporarily reduces the congestion window, facilitates periodic bandwidth redistribution, benefiting flows with longer RTTs. This mechanism could explain the observed throughput oscillations among flows, mainly how flows with 300ms and 400ms RTTs occasionally achieve higher or similar throughput compared to those with shorter RTTs.

Such oscillations likely result from **BBR2**'s efforts to ensure fairness among streams by adjusting to the corrected RTTprop, a vital metric for the algorithm's optimized functioning in virtualized environments with temporal multiplexing. By dynamically adjusting the sending rate and CWND based on these metrics, **BBR2** can counteract the effects of inflated RTT due to virtualization, allowing for proactive adaptation to bandwidth availability and congestion levels.

A similar behavior emerges with **CUBIC**, where its CWND also enters a period of adjustment. However, **CUBIC**'s adjustment process appears slower, gradually responding to changes observed in the network. For example, as Figure 5.13 shows, **CUBIC** starts adjusting between 20s and 30s into the experiment, while **BBR2** initiates this process as early as 10s. Despite this slower initial response, **CUBIC** maintains performance with less fluctuation over time, achieving slightly better and more stable fairness values. This behavior indicates that although **CUBIC** responds to network changes less promptly than **BBR2**, its ability to sustain consistent performance and fairness in bandwidth distribution becomes apparent once the network stabilizes. This behavior can be explained by the fact that **CUBIC** employs a cubic growth function for its CWND to optimize long-term network throughput, allowing for a more gradual increase in bandwidth usage after a congestion event.

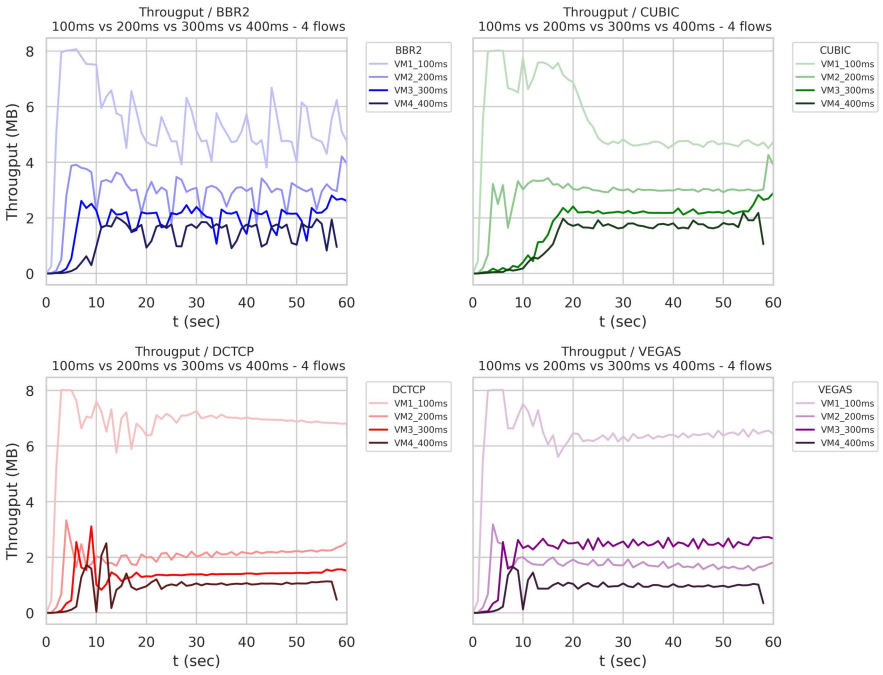
3. **Retransmissions**

   - **BBR2** is the only algorithm with no packet retransmission during the entire experiment for any flow.

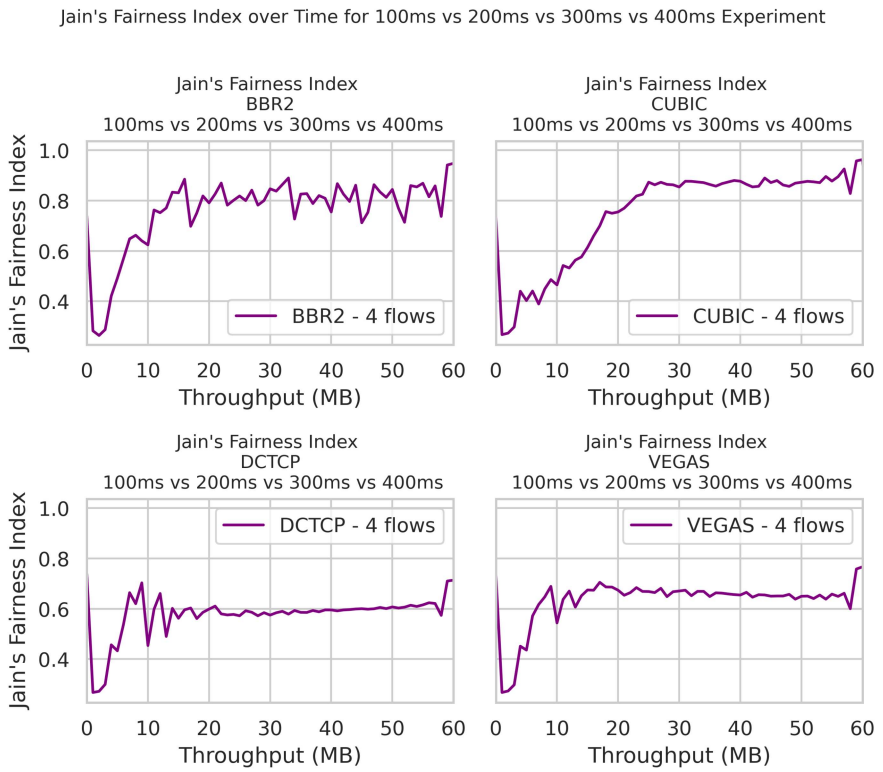
Concluding Remarks

The insights from the "Four-Level Delayed Flow Analysis" experiment highlight the importance of algorithm adaptability in environments with fluctuating network conditions. The results indicate that both **BBR2** and **CUBIC** are potentially more adaptable to environments with variable network latency, as evidenced by their throughput fairness over time. The CWND adaptation mechanism of **BBR2** significantly affects its ability to handle latency fluctuations effectively, maintaining a balanced throughput among competing flows. However, it is worth noting that **BBR2**'s throughput is inconsistent, exhibiting slight variation. On the other hand, **CUBIC**'s gradual attainment of fairness suggests that while it may take longer to adapt to changes in latency, it eventually achieves a comparable level of throughput fairness. These findings underscore the potential suitability of **BBR2** and **CUBIC** for networks where latency can vary significantly over time, providing valuable guidance for network engineers in designing and managing TCP congestion control mechanisms within data centers and virtualized settings.

It is crucial for future network infrastructure planning to consider the adaptability of congestion control algorithms to latency variability. This experiment provides a foundation for such considerations, offering a comparative perspective on how different algorithms perform under changing network latency.



(A) Throughput for algorithms in Delayed with Four Flows scenario



(B) Fairness for algorithms in Delayed with Four Flows scenario

Figure 5.13: Throughput (A) and Fairness (B) for Delayed with Four Flows scenario.

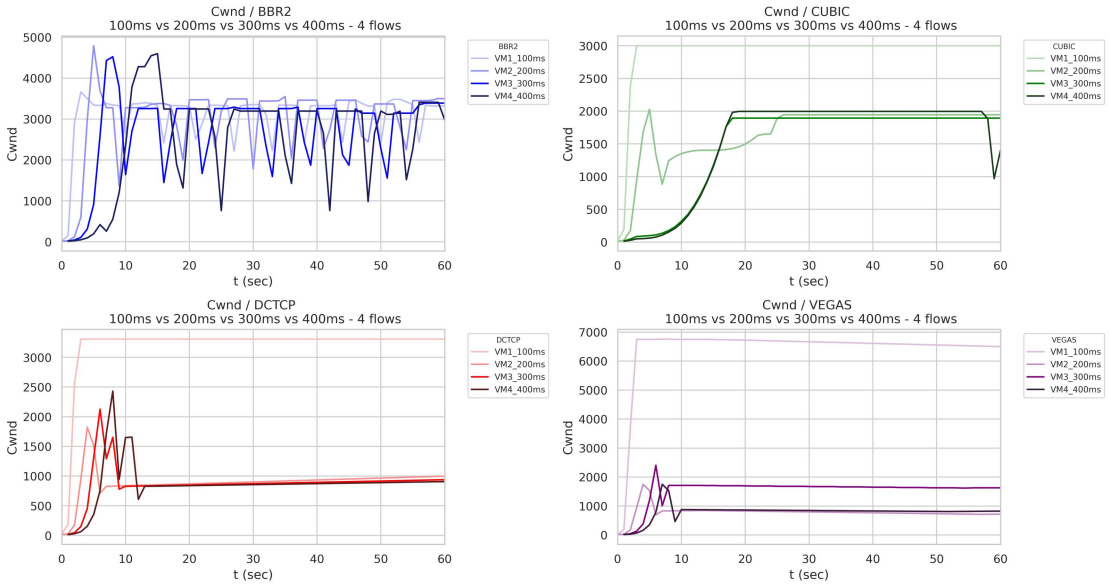


Figure 5.14: CWND (KBytes) for algorithms in Delayed with Four Flows scenario.

5.4 Basic Network Failures Scenario: Packet Loss Impact on Single VM

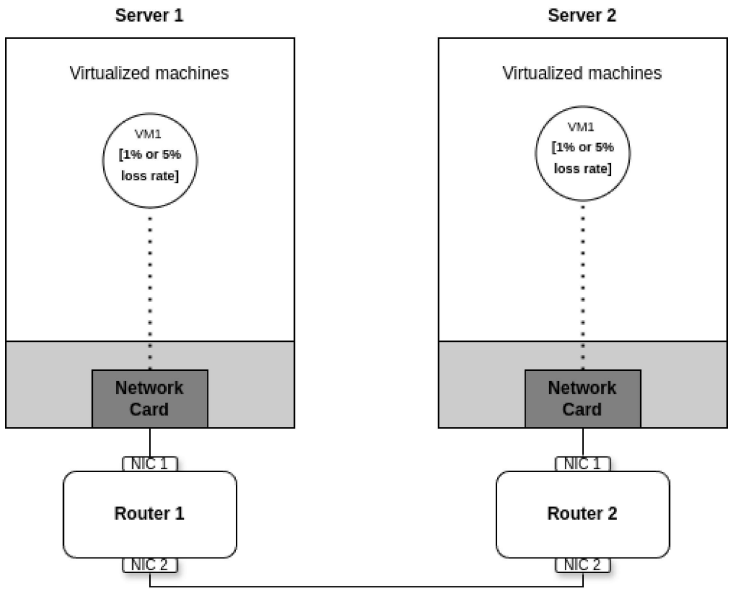


Figure 5.15: Network setup for the “Packet Loss Impact on Single VM” scenario.

This section examines the resilience of the congestion control algorithms to packet loss scenarios by assessing their performance in a Single VM setup under two different packet loss rates: 1% and 5%. The experiment lasted 60 seconds to understand the packet loss’s impact on each algorithm’s network performance. The summary of results for this scenario is found in Table 5.4. Figure 5.15 represents the network setup for this scenario.

Table 5.4: Comparative Analysis of TCP Congestion Algorithms in “Packet Loss Impact on Single VM" Scenario

Number of VMs	VM Loss (%)	Congestion Algorithm	Average Cwnd (KBytes)	Total Retransmissions	Average RTT (ms)	Average Throughput per Flow (MB)	Total Data Sent (MB)	Average Sending Rate per flow (MBytes/sec)
1	1%	BBR2	5105.134	4929	5.0	11.070	675.300	10.868
	5%	BBR2	1147.440	21475	1.0	9.849	600.785	9.217
	1%	CUBIC	818.556	4558	1.0	10.441	636.891	10.189
	5%	CUBIC	239.272	4227	4.0	1.941	118.381	1.794
	1%	DCTCP	1186.811	4967	1.0	11.005	671.329	10.775
	5%	DCTCP	307.123	6811	2.0	3.101	192.243	2.929
	1%	VEGAS	607.649	4171	1.0	9.588	584.889	9.319
	5%	VEGAS	239.518	4088	2.0	1.935	118.042	1.746

1. **Sending Rate Observations**

   - **BBR2** demonstrated higher average sending rates, maintaining efficient data transfer even under packet loss conditions, while **VEGAS** showed the lowest rates.
   - **CUBIC**, **DCTCP**, and **VEGAS** experienced a notable decrease in average sending rates when the packet loss increased from 1% to 5%, in contrast to **BBR2**, which exhibited a less significant decrease.

2. **Throughput and Fairness**

   - As for the sending rate, **BBR2** shows higher throughput rates than the others, indicating a high throughput rate even under loss conditions. **VEGAS** has the lowest throughput rates as shown in Figure 5.16.

3. **Congestion Window (CWND)**

   - **BBR2** managed to sustain a considerably larger congestion window than other algorithms at a packet loss rate of 1%. This indicates it can discern between losses caused by network congestion or other factors. Even at a 5% loss rate, **BBR2** maintained a larger CWND, though with a higher loss rate, compared to its peers. **BBR2** estimates bandwidth; for this, it can maintain a higher cwnd. See Figure 5.17.

4. **Round-Trip Time (RTT)**

   - All algorithms maintained a low and stable RTT.

Concluding Remarks

The "Single VM packet loss" experiments reveal that **BBR2** stands out in terms of resilience to packet loss, maintaining higher throughput and a more stable RTT than other algorithms. The findings suggest that **BBR2** is potentially more adept at distinguishing between congestion-induced losses and losses caused by other factors. In contrast, **VEGAS** struggles with lower throughput and sending rates. **CUBIC** and **DCTCP**, while generally stable at a 1% loss rate, show signs of instability at a 5% rate of packet loss.

These results are instrumental for network designers in selecting TCP congestion control algorithms that can deliver consistent performance in the presence of packet loss, a common challenge in virtualized environments.

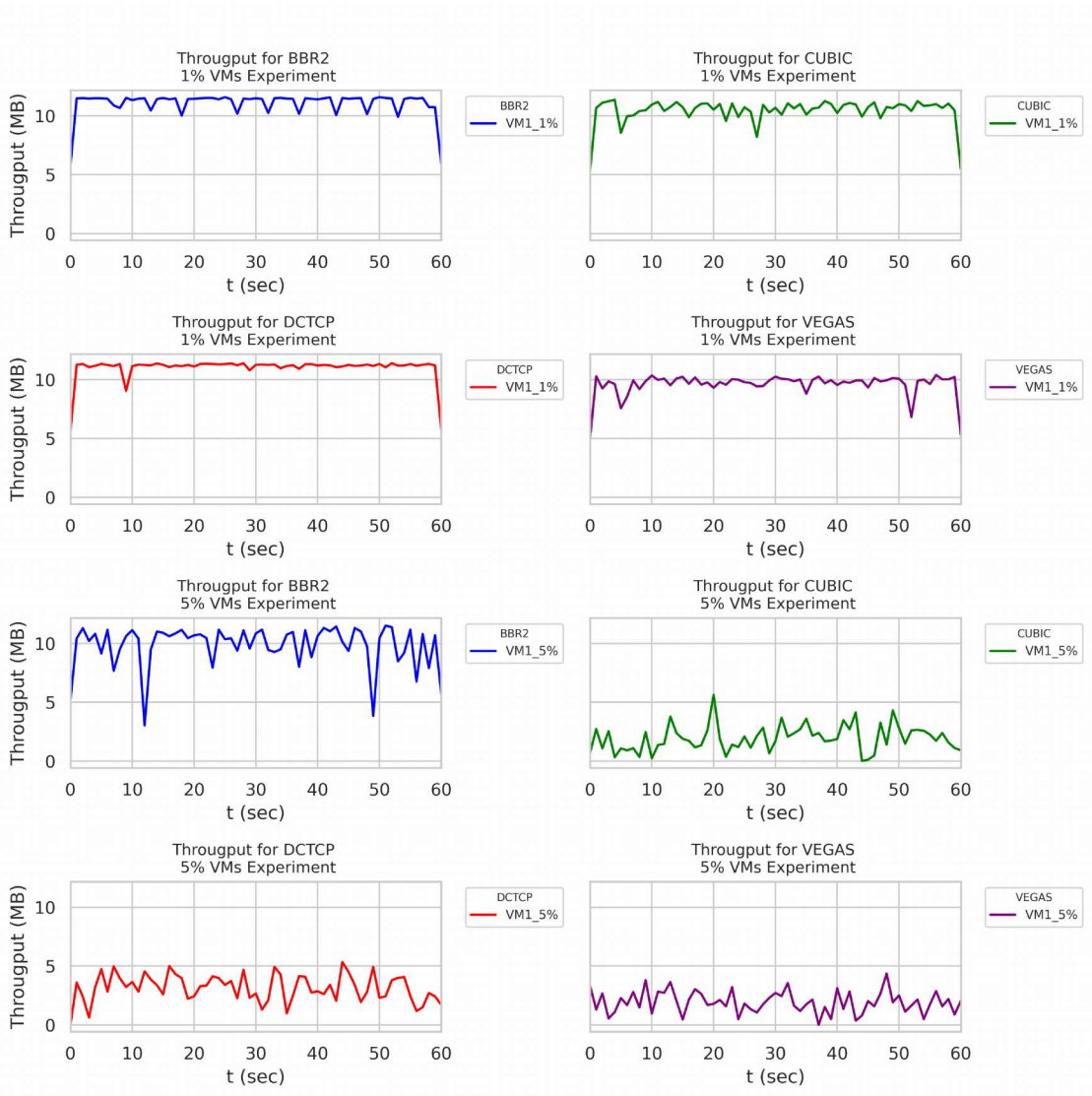


Figure 5.16: Throughput for algorithms in Packet Loss Single VM scenario.

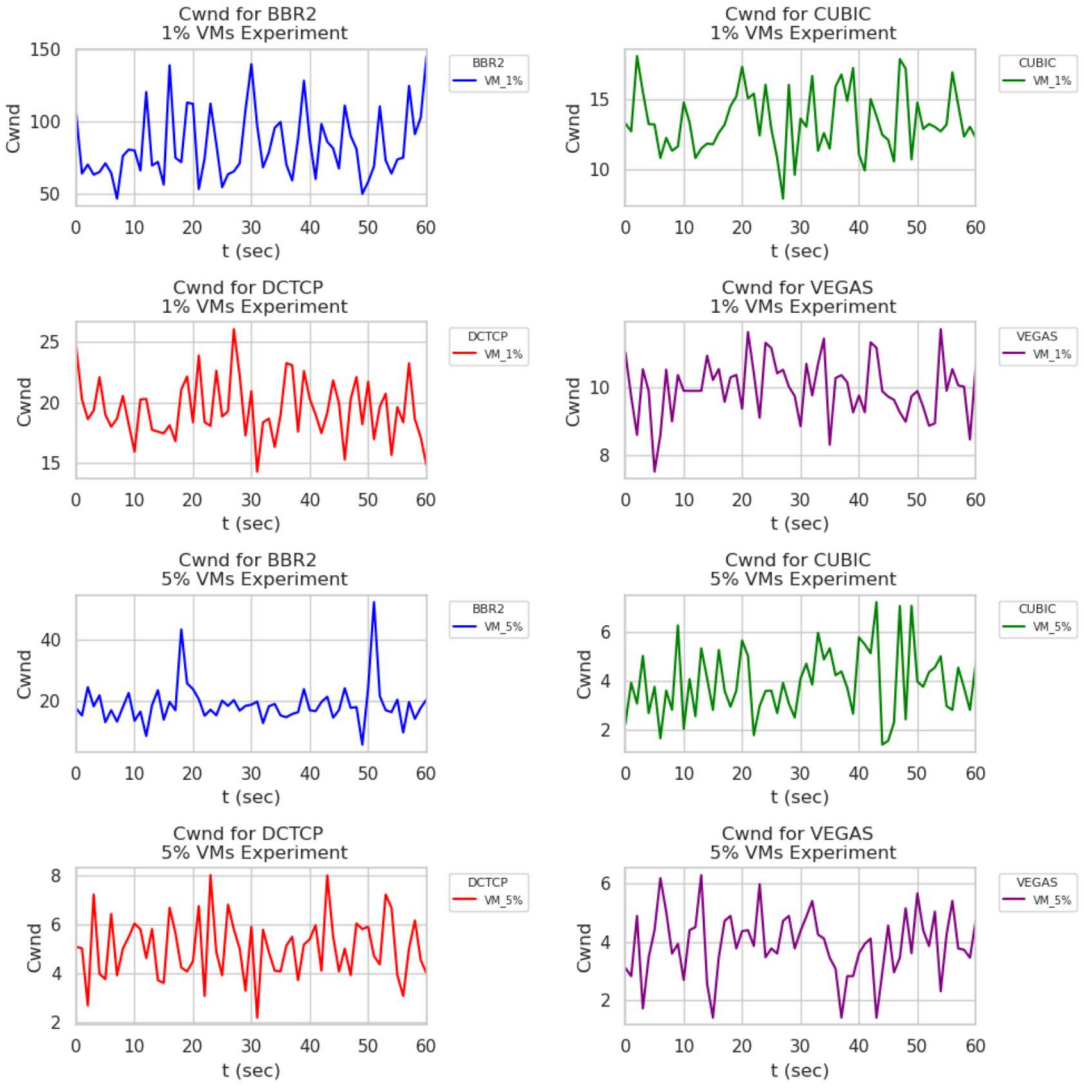


Figure 5.17: CWND (KBytes) for algorithms in Packet Loss Single VM scenario.

5.5 Basic Network Failures Scenario: Packet Loss Impact in Dual-VM Setup

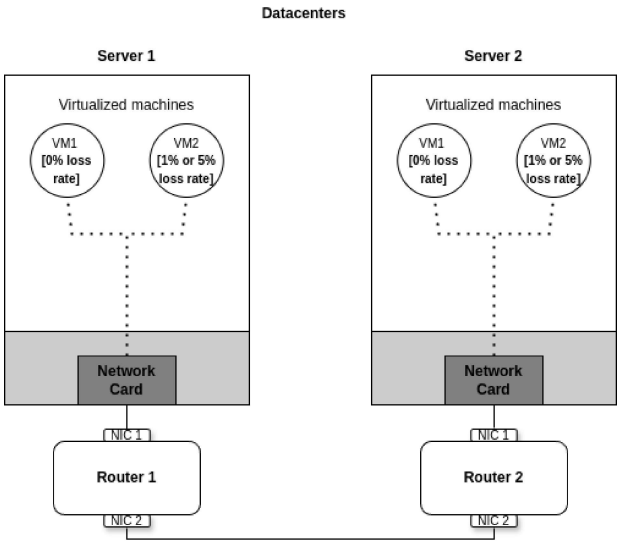


Figure 5.18: Network setup for the “Packet Loss Impact in Dual-VM Setup” scenario.

Figure 5.18 represents the network setup for this scenario.

The objective of this scenario is to evaluate the behavior of congestion control algorithms when one VM experiences packet loss while another operates under normal conditions. This scenario helps identify how the algorithms manage uneven network quality between flows. The summary of results for this scenario is found in Table 5.5.

Table 5.5: Comparative Analysis of TCP Congestion Algorithms in “Packet Loss Impact in Dual-VM Setup" Scenario

Number of VMs	VM Number	VM Loss (%)	Congestion Algorithm	Average Cwnd (KBytes)	Total Retransmissions	Average RTT (ms)	Average Throughput per Flow (MB)	Total Data Sent (MB)	Average Sending Rate per flow (MBytes/sec)	Average Fairness Index
2	VM1 VM2	0% 1%	BBR2	10504.784 4378.351	0 1441	18.0 18.0	8.132 3.259	496.065 198.779	8.114 3.217	0.84
	VM1 VM2	0% 5%	BBR2	10205.913 1318.408	0 2850	14.0 16.0	10.032 1.303	611.960 79.490	10.008 1.244	0.63
	VM1 VM2	0% 1%	CUBIC	102865.649 1109.171	0 103	138.0 138.0	11.192 0.241	682.697 14.727	11.149 0.239	0.52
	VM1 VM2	0% 5%	CUBIC	102917.850 415.935	0 181	137.0 146.0	11.355 0.082	692.674 5.066	11.218 0.079	0.52
	VM1 VM2	0% 1%	DCTCP	201136.128 1200.122	0 171	141.0 139.0	11.276 0.371	687.845 23.015	11.143 0.375	0.53
	VM1 VM2	0% 5%	DCTCP	201138.176 478.007	0 324	141.0 150.0	11.259 0.137	686.790 8.612	11.216 0.132	0.53
	VM1 VM2	0% 1%	VEGAS	586.460 536.584	0 2228	1.0 1.0	6.262 5.251	381.971 320.330	6.152 5.098	0.99
	VM1 VM2	0% 5%	VEGAS	566.186 240.051	0 3396	1.0 2.0	9.637 1.506	587.853 91.867	9.470 1.391	0.65

1. **Sending Rate Observations**

   - **VEGAS** and **BBR2** showed more balanced sending rates between flows with different loss rates. Specifically, with 1% loss, **VEGAS** maintained nearly identical sending rates for both flows. With loss increased to 5%, both **VEGAS** and **BBR2** exhibited an imbalance in sending rates but still outperformed **DCTCP** and **CUBIC** in terms of sending rate distribution equity. See column "Average Sending Rate per flow" in Table 5.5.

2. **Throughput and Fairness**

   - **VEGAS** and **BBR2** maintained the best throughput fairness indices. With 1% loss, **VEGAS** almost reached the maximum fairness index. When the loss was increased to 5%, the fairness indices of **VEGAS** and **BBR2** worsened but remained better than the other algorithms. In the case of **VEGAS**, this worsening is much more related to the fact that packet loss caused an increase in the RTT of the flow with 5% losses than related to the loss itself since **VEGAS** is a delay-based algorithm. See Figure 5.19 and 5.20.

3. **Congestion Window (CWND)**

   - **DCTCP** and **CUBIC** assign high CWND values to lossless flows (0% loss), reaching approximately 3000 for **DCTCP** and 1500 for **CUBIC**, regardless if the other flows have 1% or 5% loss. Conversely, **VEGAS** and **BBR2** allocate considerably lower values to the lossless flow, with **BBR2** around 175 and **VEGAS** at 12. This approach results in a closer CWND value range between 0%, 1%, and 5% flows for **VEGAS** and **BBR2**. See Figure 5.21.

4. **Round-Trip Time (RTT)**

   - For 1% and 5% losses, **BBR2** and **VEGAS** maintained RTTs below 20ms, while **DCTCP** and **CUBIC** reached RTTs close to 140ms.

5. **Retransmissions**

   - **BBR2** and **VEGAS** had more retransmissions, with 1441 and 2228 packets for 1% loss and 2850 and 3396 for 5%, respectively. **CUBIC** and **DCTCP** suffered significantly fewer retransmissions, with **CUBIC** retransmitting 103 packets for 1% and 181 for 5% packet loss whereas **DCTCP** retransmitted 171 packets for 1% and 324 for 5% loss level. These numbers reflect the tendency of **CUBIC** and **DCTCP** not to allow packet-loss flows to access available bandwidth. See Figure 5.22.

Concluding Remarks

The "Packet Loss Impact in Dual-VM Setup" experiment underlines the adaptability and performance of **VEGAS** and **BBR2** in handling packet loss disparities across different flows. **VEGAS** and **BBR2** proved more efficient in managing packet loss, with better balance in sending rate and higher throughput fairness indices. Both maintain relatively high throughput rates and low RTTs even under significant loss conditions, suggesting superior robustness in such scenarios. Despite a general decline in performance as packet loss increased, **VEGAS** and **BBR2** maintained superior throughput and fairness indices compared to **CUBIC** and **DCTCP**. **CUBIC** and **DCTCP**, although exhibiting fewer retransmissions, showed inferior performance in terms of fairness, which may be a disadvantage in networks with variable packet loss.

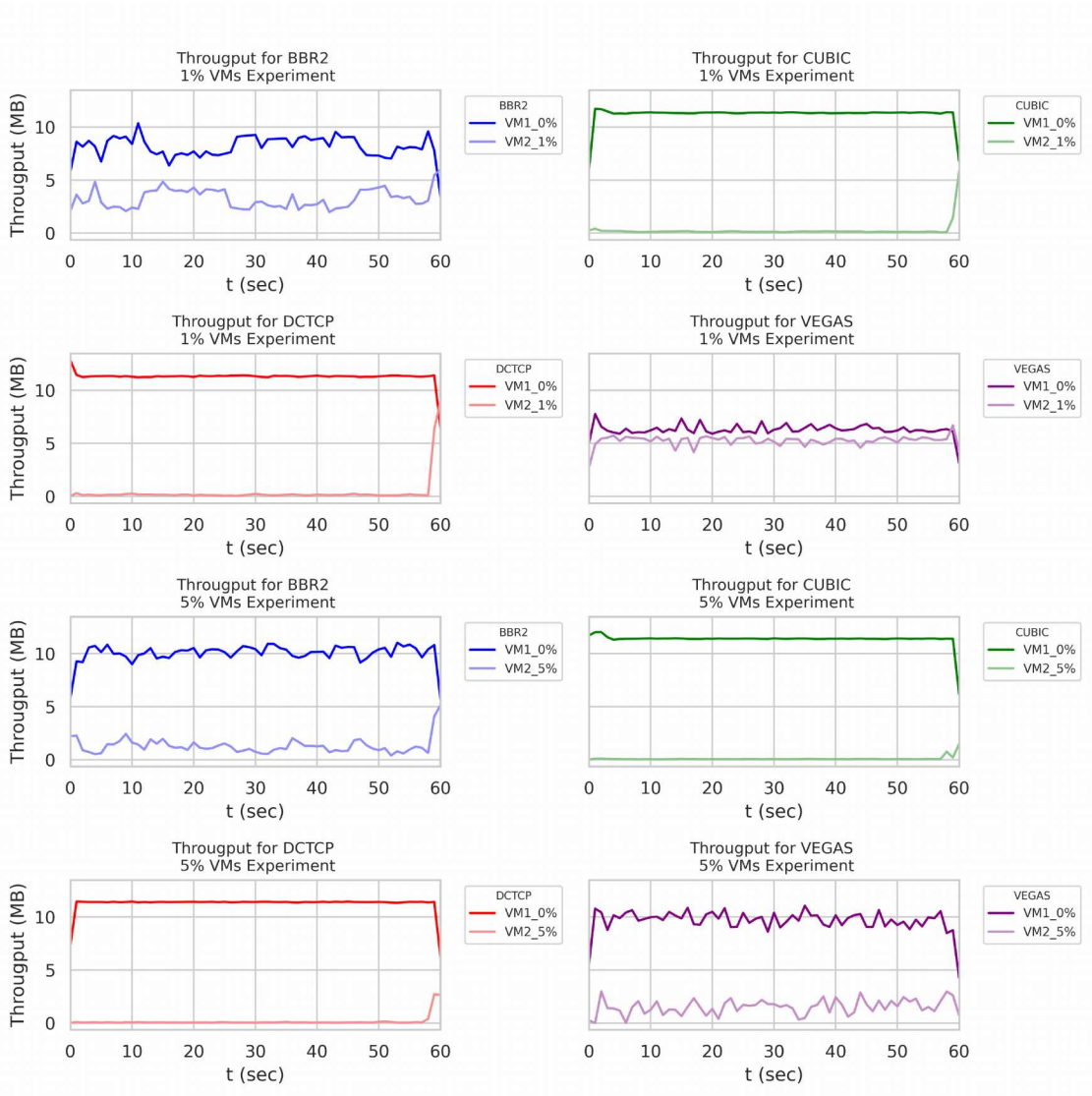


Figure 5.19: Throughput for algorithms in Packet Loss in Dual-VM Setup scenario.

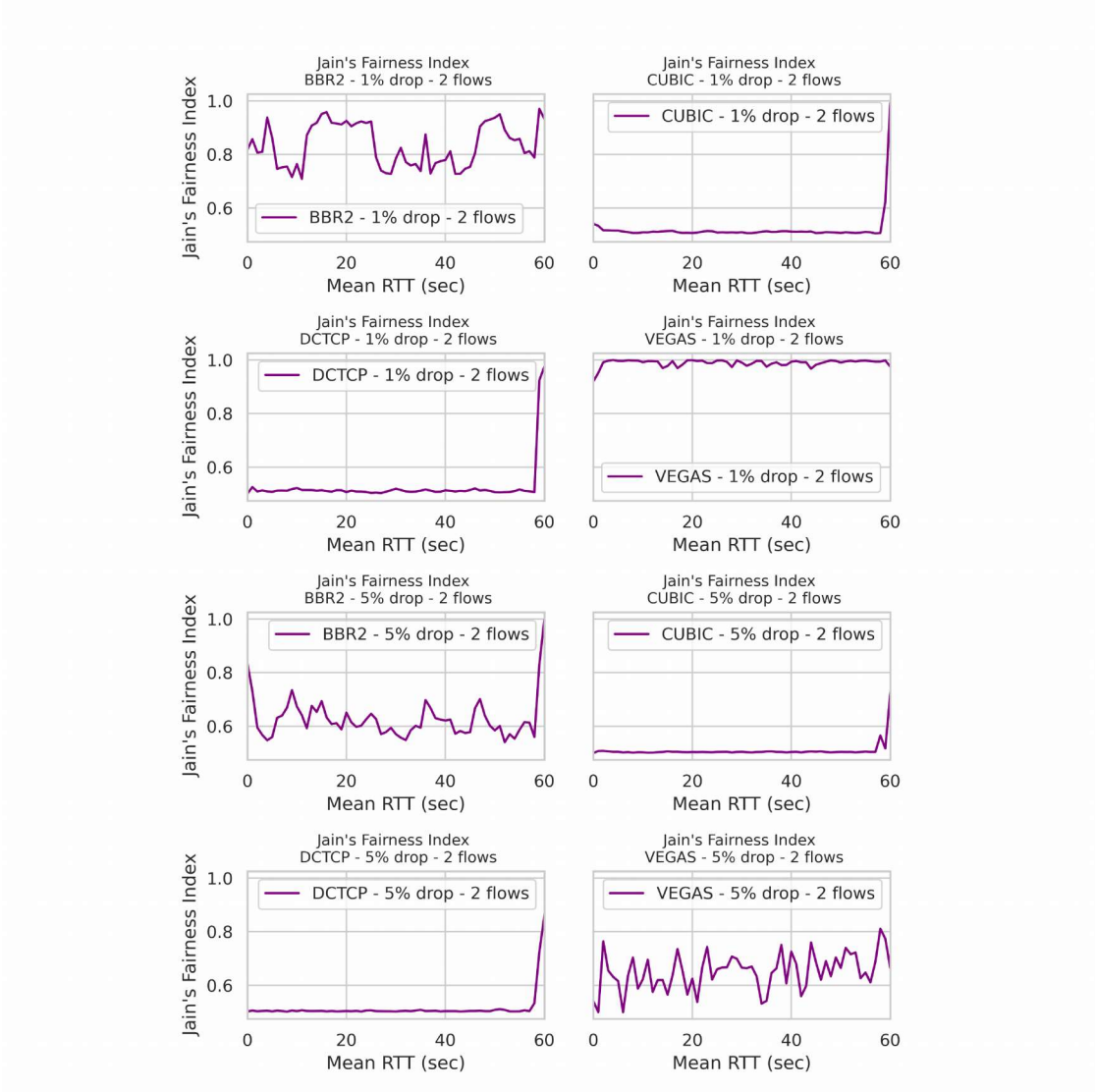


Figure 5.20: Fairness for algorithms in Packet Loss in Dual-VM Setup scenario.

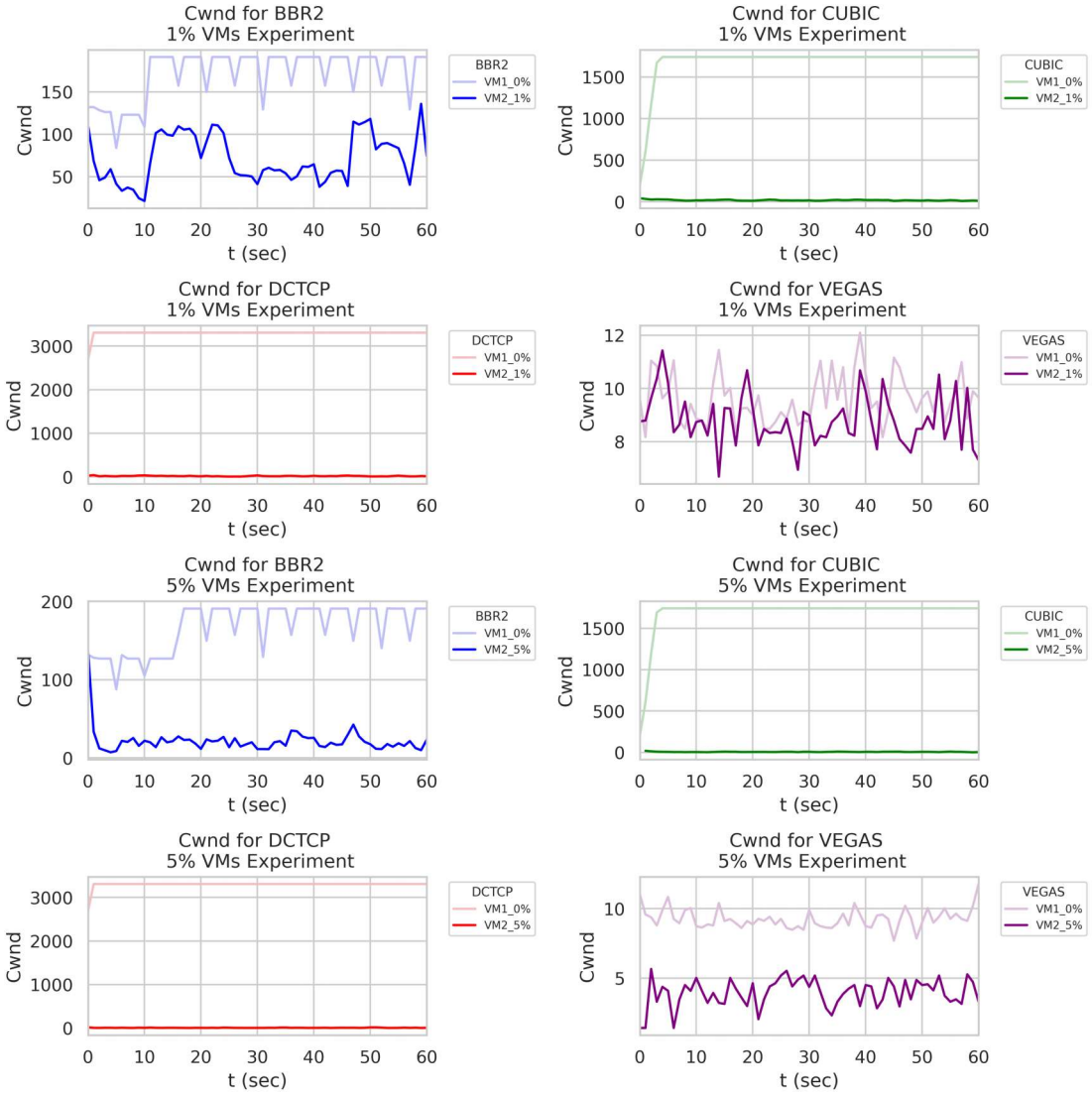


Figure 5.21: CWND (KBytes) for algorithms in Packet Loss in Dual-VM Setup scenario.

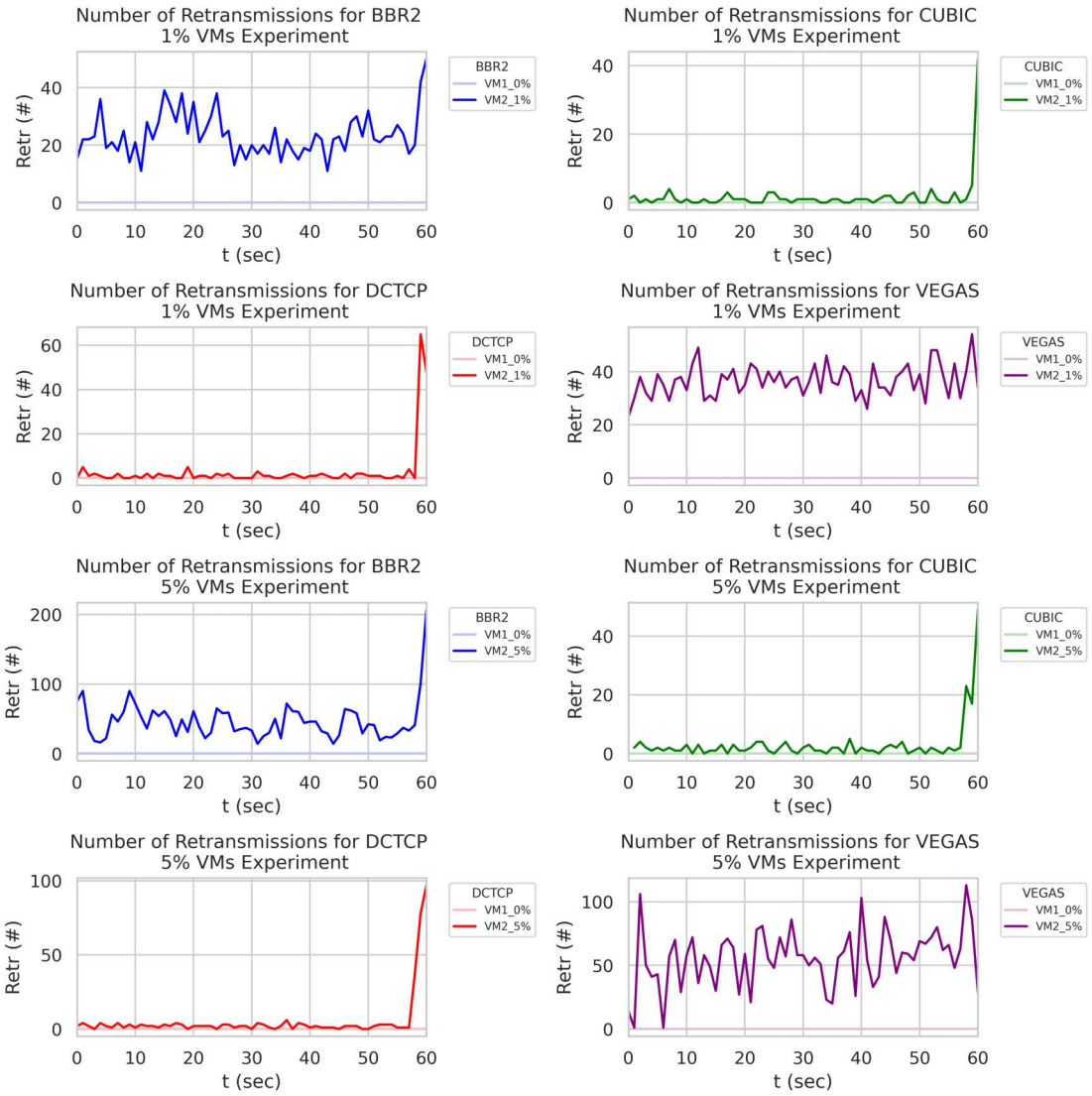


Figure 5.22: Retransmission for algorithms in Packet Loss in Dual-VM Setup scenario.

5.6 Multiple: Dual-Algorithm Competition (2 VMs)

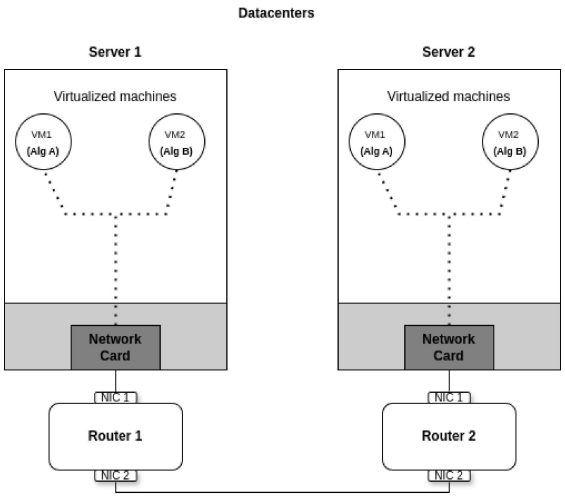


Figure 5.23: Network setup for the “Dual-Algorithm Competition (2 VMs)” scenario.

In the “Multiple Algorithms Scenario” group, the “Dual-Algorithm Competition (2 VMs)” experiment was designed to assess how different TCP congestion control algorithms perform when directly competing for network resources. This setup involved two virtual machines running different congestion control algorithms under identical network conditions. This configuration aimed to reveal how these algorithms manage bandwidth and network performance amidst direct competition with other algorithms. The summary of results for this scenario is found in Table 5.6. Figure 5.23 represents the network setup for this scenario.

Table 5.6: Comparative Analysis of TCP Congestion Algorithms in ‘Multiple: Dual-Algorithm Competition" Scenario

Number of VMs	VM Number	Congestion Algorithm	Average Cwnd (KBytes)	Total Retransmissions	Average RTT (ms)	Average Throughput per Flow (MB)	Total Data Sent (MB)	Average Sending Rate per flow (MBytes/sec)	Average Fairness Index
2	VM0	BBR2	43469.10	0	174.00	2.361	146.361	2.358	0.74
	VM1	DCTCP	200625.76	0	177.00	9.10	555.127	9.06	
	VM0	CUBIC	102393.10	0	202.00	7.729	479.185	7.807	0.86
	VM1	BBR2	74333.77	0	207.00	3.64	222.051	3.633	
	VM0	CUBIC	98509.29	0	270.00	5.721	354.689	5.754	0.98
	VM1	DCTCP	198323.90	0	275.00	5.731	349.562	5.708	
	VM0	CUBIC	102850.97	0	142.00	10.88	663.695	10.835	0.56
	VM1	VEGAS	4188.07	1	141.00	0.584	35.608	0.577	
	VM0	VEGAS	1431.03	1	8.00	2.868	174.928	2.817	0.80
	VM1	BBR2	5556.22	0	9.00	8.561	522.218	8.55	
	VM0	VEGAS	4434.88	0	143.00	0.613	38.017	0.615	0.56
	VM1	DCTCP	201105.41	0	147.00	10.846	661.591	10.806	

1. **Sending Rate and Throughput**

   - **DCTCP** and **CUBIC** dominated the link capacity by sending more data when competing against **VEGAS** and **BBR2**, with a more pronounced dominance over **VEGAS**. When **DCTCP** and **CUBIC** competed against each other, their performance was quite similar, displaying data transmission rates close to each other and achieving a fairness index near 1. **BBR2** sends more data only when disputes with **VEGAS** as depicted in Figure 5.24.

2. **Congestion Window (CWND)**

   - Both **DCTCP** and **CUBIC** achieved larger CWND sizes when competing against other algorithms, indicating their aggressive approach toward bandwidth utilization. **DCTCP**, in particular, displayed the largest CWND sizes, even when matched against **CUBIC**, suggesting its slightly more assertive stance in claiming network capacity. However, against **CUBIC**, **DCTCP** competed more equally regarding congestion window size than against **VEGAS** and **BBR2**. See Figure 5.26.
   - An interesting dynamic was observed between **CUBIC** and **BBR2**. **BBR2** started at a disadvantage with a smaller CWND but saw its window size increase as the experiment progressed. By the end, **BBR2** was able to compete on par with **CUBIC**. A similar pattern emerged in the **BBR2** *vs*. **DCTCP** competition, albeit more gradually, due to **DCTCP**'s larger CWND sizes and its more aggressive behavior. See Figure 5.26.

3. **Round-Trip Time (RTT)**

   - Low and stable Round-Trip Time (RTT) values for virtual machines (VMs) are observed during the experiment, regardless of the algorithm used.
   - **VEGAS** VS. **BBR2**: Notably, this combination features the lowest RTT (8 and 9 milliseconds), significantly lower than all other combinations, indicating a highly efficient interaction between VEGAS and BBR2, resulting in minimal latencies.
   - The interaction between **VEGAS**, **BBR2**, and the other algorithms highlights a dichotomy in congestion control: **VEGAS** and **BBR2** maintain low RTTs when operating together, demonstrating an efficient synergy. However, faced with adversaries such as **CUBIC** or **DCTCP**, whose aggressive congestion window widening strategies or reliance on packet loss detection challenge their methodologies, both face difficulties. Adapting to emulate loss-based behaviors in environments dominated by such algorithms, **VEGAS** sees its performance degraded. Similarly, **BBR2**, designed to optimize

bandwidth and RTT without relying strictly on packet losses, may compromise efficiency in these adverse conditions. In **BBR2**, this can be seen in the increase in the congestion window when competing with CUBIC and **DCTCP**. This incompatibility results in longer queues and increases in RTT, highlighting the significant impact of the congestion control strategy on network dynamics. On the other side, when VEGAS and **BBR2** operate together, they can mutually benefit from their control approaches: **VEGAS**, by keeping RTTs low through proactive congestion prevention, creates a network environment that allows **BBR2** to operate efficiently within capacity limits available without inducing packet losses that could result from an overloaded network.

Concluding Remarks

These findings highlight the distinct strategies and capabilities of congestion control algorithms in managing network resources under competition. **DCTCP** and **CUBIC**'s aggressive tactics allow them to outperform against the more conservative **VEGAS** and **BBR2**. However, the adaptability of **BBR2**, demonstrated by its ability to recover and compete effectively as the experiment progresses, showcases the nuanced balance between aggression and efficiency in network resource management.

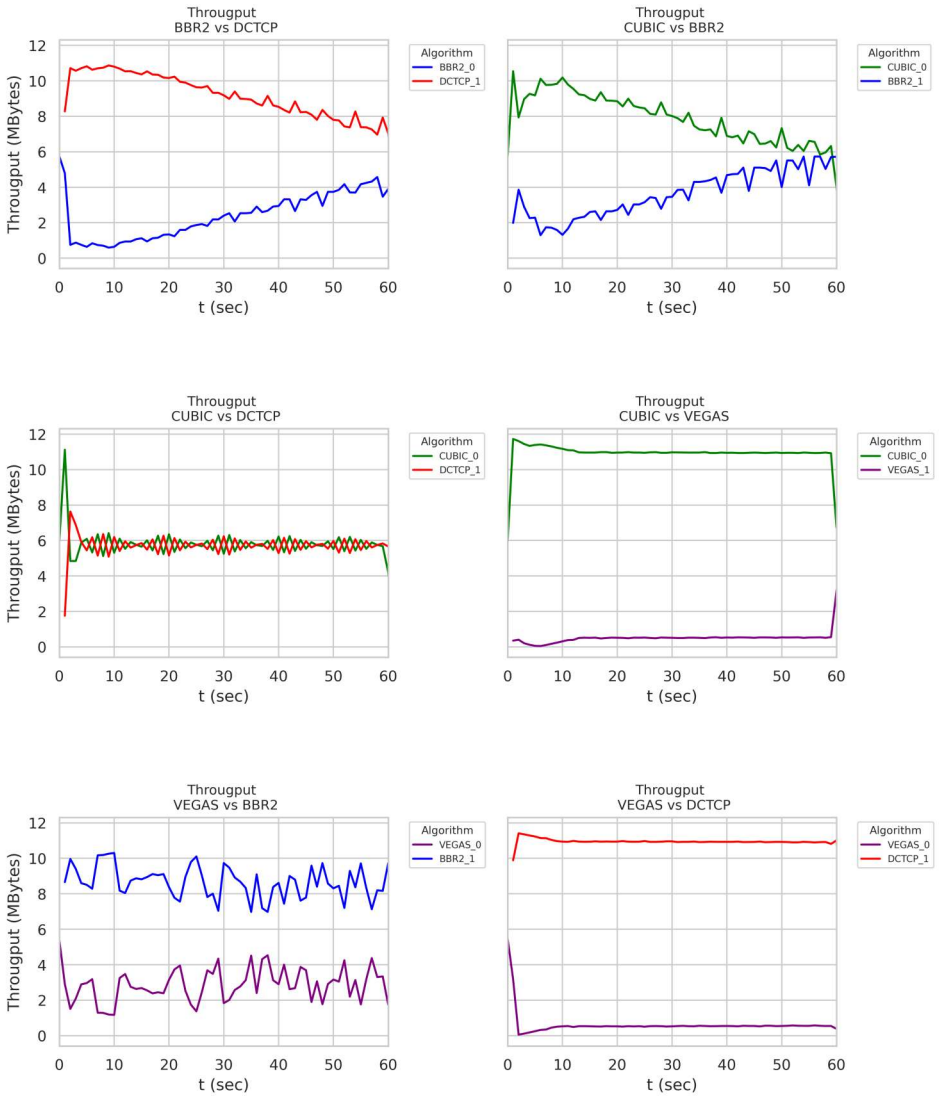


Figure 5.24: Throughput for algorithms in Dual-Algorithm Competition scenario

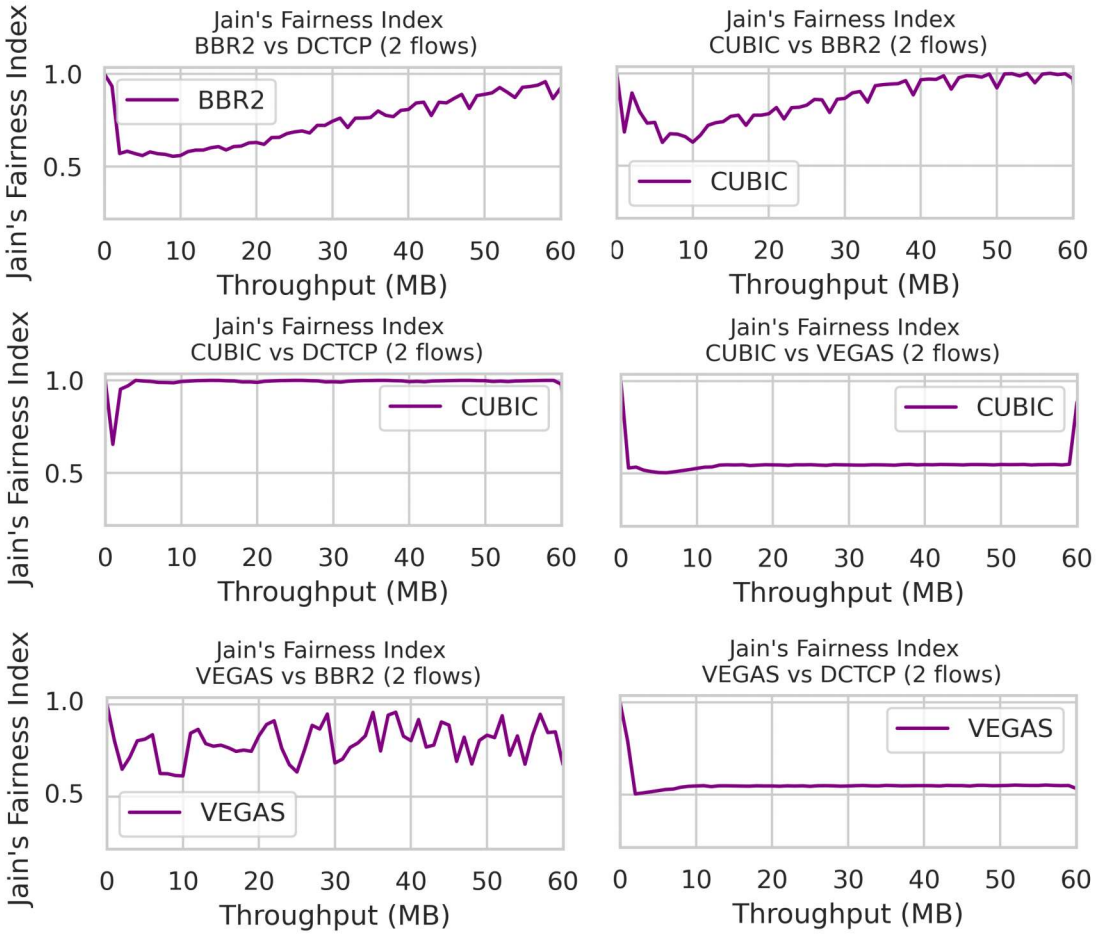


Figure 5.25: Fairness for algorithms in Dual-Algorithm Competition scenario

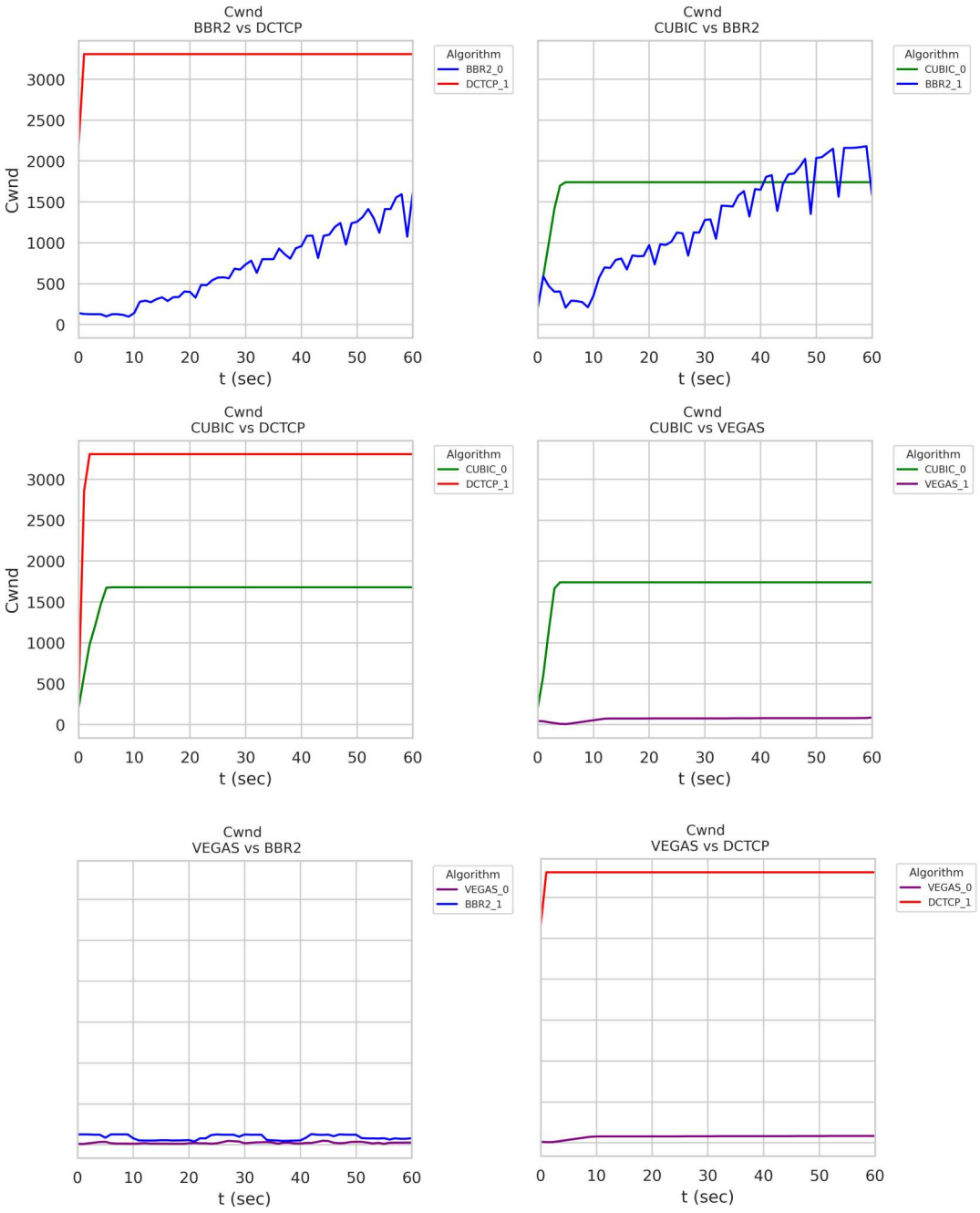


Figure 5.26: CWND (KBytes) for algorithms in Dual-Algorithm Competition scenario.

5.7 Multiple: Asymmetric Competition (4 VMs)

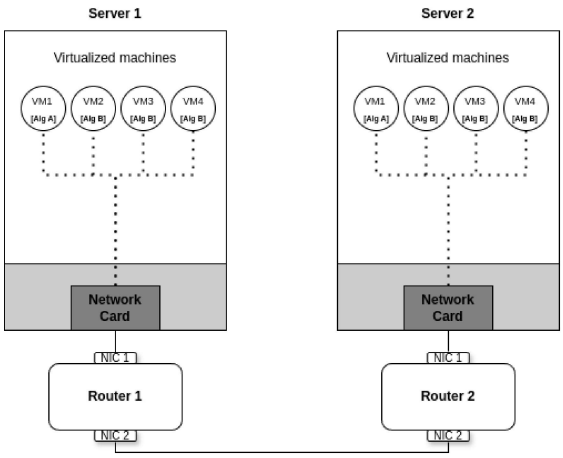


Figure 5.27: Network setup for the "Asymmetric Competition (4 VMs)" scenario.

In the "Multiple Algorithms Scenario" group, the "Asymmetric Competition (4 VMs)" experiment evaluated how different congestion control algorithms perform in an uneven competitive environment. This setup involved one VM operating with a congestion control algorithm. At the same time, the other three VMs used another algorithm, creating an asymmetric competition to assess performance and fairness when network resources are distributed unevenly. The summary of results for this scenario is found in Table 5.7. Figure 5.27 represents the network setup for this scenario.

1. **BBR2 as the Solo Algorithm**

   - When **BBR2** is the single flow used, it is at a small disadvantage regarding bandwidth usage and sending rate compared to **3xCUBIC** and **3xDCTCP** flows. Its fairness index (throughput) approaches 1 over time. Against **3xVEGAS**, **BBR2** has the advantage, with the fairness index fluctuating significantly between values below and above 0.5. See Figure 5.29.

2. **CUBIC as the Solo Algorithm**

   - **CUBIC**, when competing as the single flow against **3xBBR2**, maintains higher throughput. Against **3xDCTCP**, one of the **3xDCTCP** flows competes closely with **CUBIC** for leadership. Against **3xVEGAS**, **CUBIC** significantly dominates, nearly monopolizing the available bandwidth. Fairness indices are close to 1 against **3xDCTCP** but drop to nearly 0 against **3xVEGAS** as shown in Figure 5.29.

Table 5.7: Comparative Analysis of TCP Congestion Algorithms in “Multiple: Asymmetric Competition" Scenario

Number of VMs	VM Number	Congestion Algorithm	Average Cwnd (KBytes)	Total Retransmissions	Average RTT (ms)	Average Throughput per Flow (MB)	Total Data Sent (MB)	Average Sending Rate per flow (MBytes/sec)	Average Fairness Index
4	VM0	BBR2	45548.12	4	393.00	1.283	80.841	1.303	0.80
	VM1	CUBIC	99267.34	40	395.00	4.304	266.873	4.312	
	VM2	CUBIC	86385.21	5	395.00	3.088	194.558	3.125	
	VM3	CUBIC	86024.33	3	400.00	3.006	186.395	3.00	
	VM0	BBR2	42891.04	20	424.00	1.119	71.648	1.129	0.83
	VM1	DCTCP	173309.16	22	436.00	3.769	233.704	3.797	
	VM2	DCTCP	165547.06	35	438.00	3.417	215.258	3.494	
	VM3	DCTCP	162086.66	13	439.00	3.433	212.846	3.454	
	VM0	BBR2	13213.43	0	22.00	8.393	511.982	8.366	0.48
	VM1	VEGAS	1390.31	0	21.00	1.063	64.842	1.043	
	VM2	VEGAS	1134.55	0	21.00	0.936	57.075	0.918	
	VM3	VEGAS	1633.88	0	21.00	1.376	83.939	1.36	
	VM1	BBR2	38110.52	4	308.00	1.574	96.028	1.573	0.76
	VM2	BBR2	61784.20	8	306.00	2.343	145.244	2.381	
	VM3	BBR2	61747.13	5	304.00	2.411	149.474	2.45	
	VM0	CUBIC	97875.18	9	301.00	5.471	339.225	5.503	
	VM0	CUBIC	96203.17	2	413.00	3.725	234.691	3.79	0.89
	VM1	DCTCP	67002.09	24	417.00	2.309	143.15	2.309	
	VM2	DCTCP	99290.08	5	423.00	3.515	221.468	3.575	
	VM3	DCTCP	58498.19	516	419.00	2.239	141.031	2.27	
	VM0	CUBIC	102279.80	0	153.00	10.165	620.094	10.121	0.35
	VM1	VEGAS	4636.95	0	154.00	0.53	32.331	0.526	
	VM2	VEGAS	991.34	0	152.00	0.193	11.941	0.195	
	VM3	VEGAS	6428.38	0	149.00	0.938	58.133	0.951	
	VM0	DCTCP	112838.98	45	329.00	5.001	310.069	5.026	0.82
	VM1	BBR2	56796.82	26	334.00	2.049	127.048	2.081	
	VM2	BBR2	73989.44	33	332.00	2.743	170.064	2.773	
	VM3	BBR2	51181.55	37	330.00	2.05	127.075	2.076	
	VM0	DCTCP	93791.76	47	387.00	2.984	190.964	3.076	0.84
	VM1	CUBIC	94679.76	2	396.00	3.597	226.628	3.654	
	VM2	CUBIC	74044.67	106	400.00	2.701	167.475	2.689	
	VM3	CUBIC	64338.77	90	395.00	2.445	154.036	2.469	
	VM0	DCTCP	201132.03	0	168.00	8.996	557.735	9.059	0.42
	VM1	VEGAS	4232.39	0	172.00	0.397	24.626	0.407	
	VM2	VEGAS	5729.03	0	170.00	0.578	35.806	0.587	
	VM3	VEGAS	16026.40	0	168.00	1.797	111.409	1.81	
	VM0	VEGAS	1082.35	0	24.00	1.624	99.088	1.596	0.83
	VM1	BBR2	4145.60	0	25.00	2.811	171.499	2.806	
	VM2	BBR2	5675.55	0	24.00	3.156	195.672	3.208	
	VM3	BBR2	9610.86	0	25.00	4.192	255.731	4.177	
	VM0	VEGAS	761.88	0	355.00	0.20	12.774	0.179	0.68
	VM1	CUBIC	105499.91	0	365.00	4.768	295.625	4.789	
	VM2	CUBIC	86693.06	0	363.00	3.377	212.749	3.432	
	VM3	CUBIC	89864.49	0	368.00	3.348	207.598	3.351	
	VM0	VEGAS	3950.73	0	400.00	0.269	16.948	0.278	0.74
	VM1	DCTCP	201113.60	0	406.00	4.057	251.521	4.107	
	VM2	DCTCP	195861.96	0	410.00	3.653	230.112	3.757	
	VM3	DCTCP	193244.78	0	415.00	3.648	226.186	3.693	

3. **DCTCP as the Solo Algorithm**

   - **DCTCP**’s behavior is similar to **CUBIC**’s. DCTCP faces disadvantages against **3xCUBIC**. Its advantage over **3xBBR2** and **3xVEGAS** is slightly less than **CUBIC**’s advantage over these algorithms as Figure 5.28 illustrates.

4. **VEGAS as the Solo Algorithm**

   - **VEGAS** faces disadvantages across all configurations, yet these disadvantages are minor when compared to scenarios where Vegas is configured as **3xVEGAS**. Against **3xBBR2**, the bandwidth is equally shared in the experiment from instants 20 s to 30, with the fairness index approaching 1. Against

3xCUBIC and **3xDCTCP**, the fairness index drops closer to 0.75, as shown in Figure 5.29.

5. **Other Observations**

   - **Round-Trip Time (RTT).** Here, the observed RTT behavior aligns with findings from the "Dual-Algorithm Competition (2 VMs)" scenario. Specifically, the VEGAS and BBR2 pairing consistently exhibits low and stable RTT values, reinforcing their effective synergy across increased competition levels with four VMs.
   - **DCTCP** consistently shows a CWND of around 3000 KBytes, much higher than other algorithms, but this drops significantly against **CUBIC** in both scenarios—when competing against three flows and facing off against a single flow. Against **1xBBR2**, the CWND drops to 1500 around 10 seconds into the experiment, and against **3xBBR2** drops to 1500 around 40 seconds. See Figure 5.30.
   - **CUBIC** maintains a CWND of around 1500 regardless of the scenario. **VEGAS** maintains lower CWND values due to its conservative nature as shown in Figure 5.30.
   - **BBR2** does not primarily adjust its congestion window to manage throughput; it estimates the available network bandwidth and adjusts its sending rate accordingly. This behavior allows **BBR2** to maintain efficient throughput even when competing with multiple Vegas flows, which adjust their congestion windows based on packet delay variations. Therefore, **BBR2**'s approach to bandwidth management can sometimes result in a lower observed CWND compared to other algorithms under similar conditions, especially against less aggressive algorithms like **VEGAS**. See Figure 5.28.

Concluding Remarks

The asymmetric competition experiment with four virtual machines (VMs) provides critical insights into the performance dynamics of TCP congestion control algorithms in environments characterized by uneven competition. Among the key findings, **BBR2**'s ability to adapt its bandwidth usage swiftly and effectively in a scenario where it is outnumbered showcases its robustness in various network conditions. With its aggressive bandwidth consumption, **CUBIC** demonstrates its capacity to optimize throughput in competitive network environments, particularly when it dominates less aggressive algorithms such as **VEGAS**.

In contrast, **VEGAS**'s performance is notably less effective in this competitive setting, pointing towards its limitations in aggressively contested network environments. This observation suggests a potential mismatch between **VEGAS**'s conservative approach and scenarios where aggressive bandwidth acquisition is beneficial.

The experiment underscores the necessity of selecting congestion control algorithms that align with the competitive landscape of the network. This choice is crucial for maintaining an equilibrium between throughput, fairness, and efficient resource utilization. Insights into the behavior of round-trip times (RTT) and congestion window (CWND) sizes further guide network administrators in optimizing network performance through strategic algorithm selection.

For instance, in a dual-algorithm competition scenario where **BBR2** and **CUBIC** coexist, **BBR2** maintains lower round-trip times (RTTs) by minimizing queuing delays. At the same time, **CUBIC** might aggressively increase its congestion window (CWND) in high-bandwidth environments, suggesting that **BBR2** is more suited for latency-sensitive applications. In contrast, **CUBIC** is advantageous when maximizing throughput is the priority. With its less aggressive approach to increasing CWND, **Vegas** is recommended for networks where maintaining low network congestion is paramount. **DCTCP** shows superior performance in data centers or environments with mixed RTTs, where its algorithm effectively reduces RTT fluctuations and stabilizes congestion window sizes, thus maintaining consistent performance levels across various traffic types. Each of these examples guides network administrators in optimizing network performance through strategic algorithm selection, emphasizing the importance of understanding the specific demands and characteristics of the network environment when choosing a congestion control strategy.

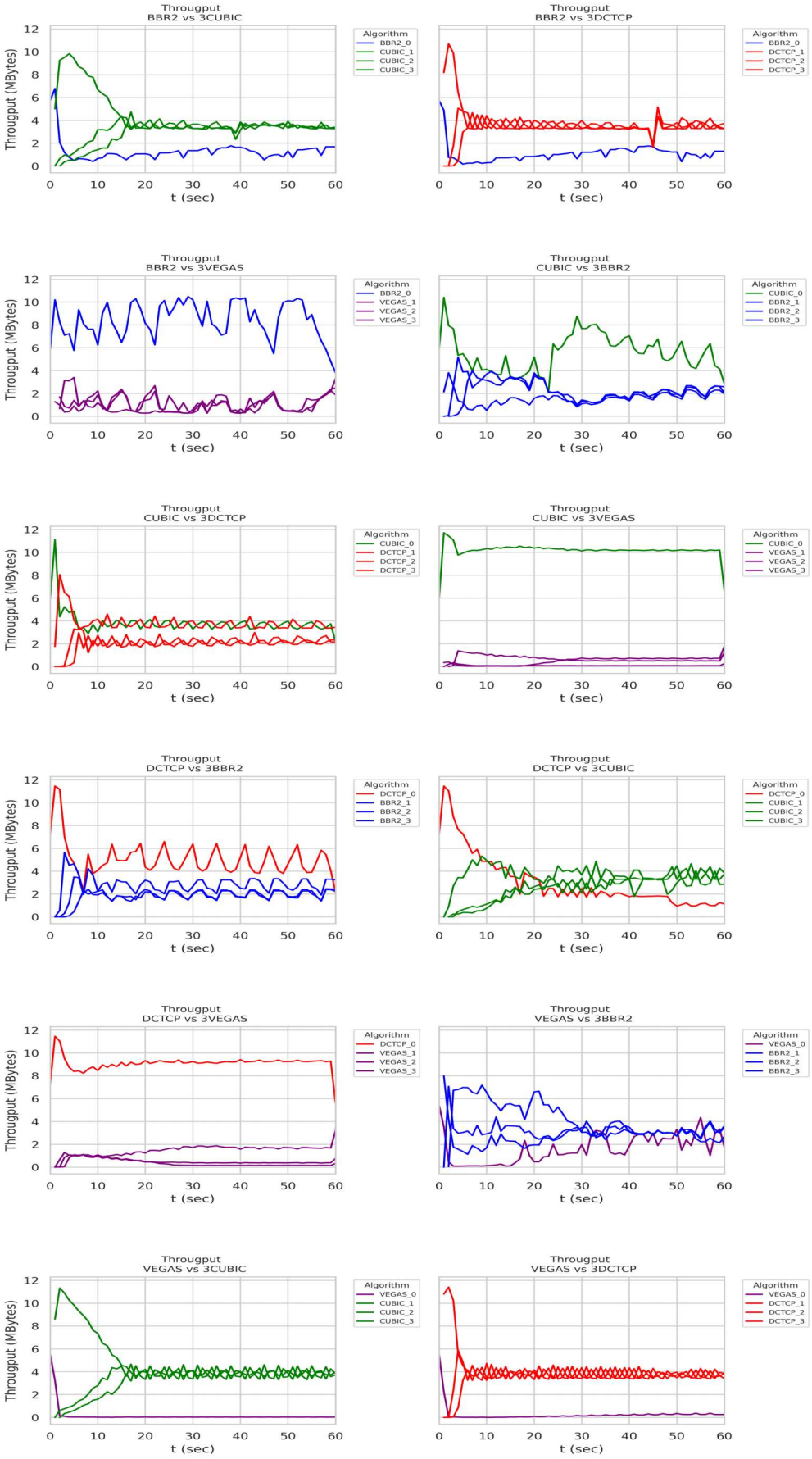


Figure 5.28: Throughput for algorithms in Asymmetric-Algorithm Competition scenario

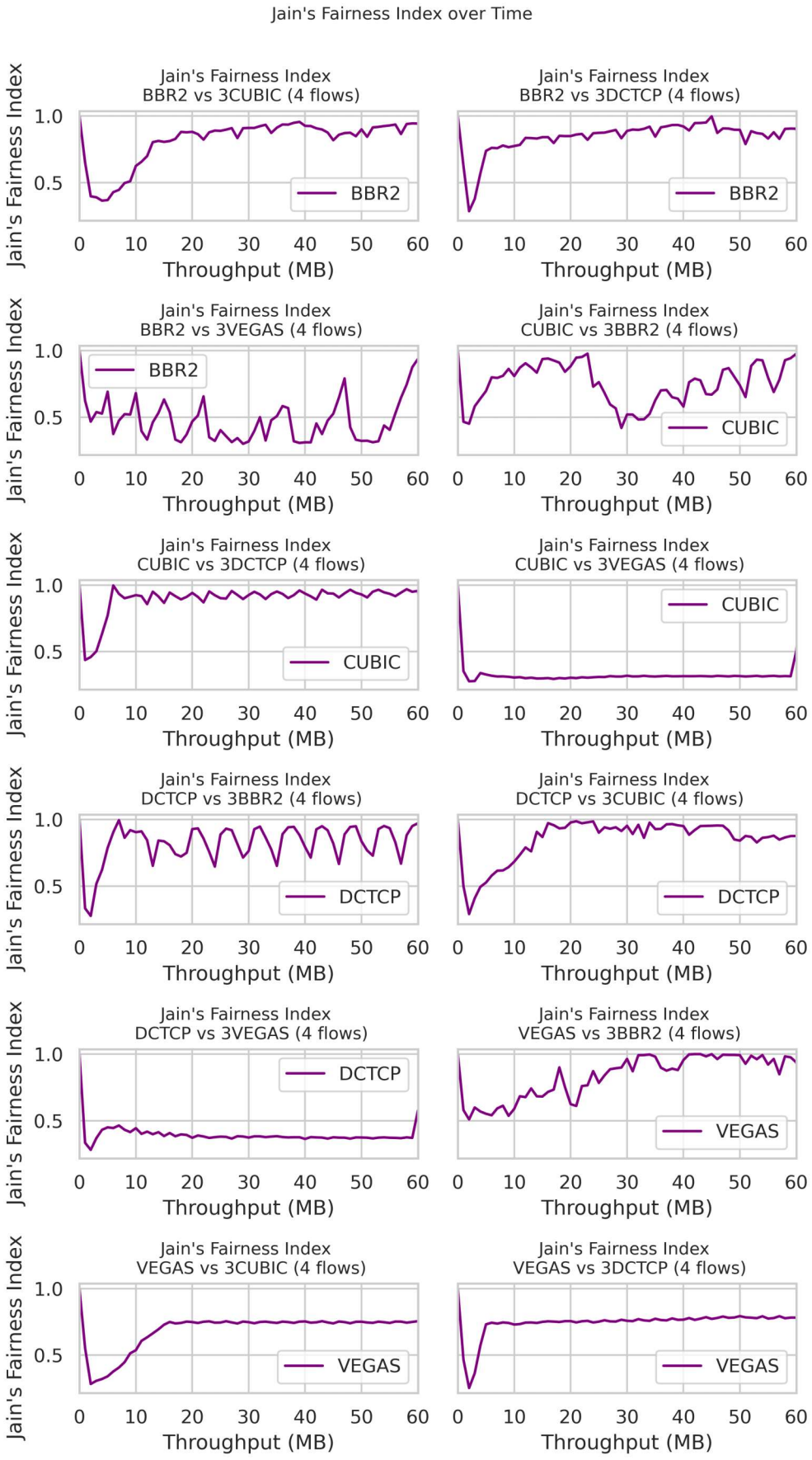


Figure 5.29: Fairness for algorithms in Asymmetric-Algorithm Competition scenario

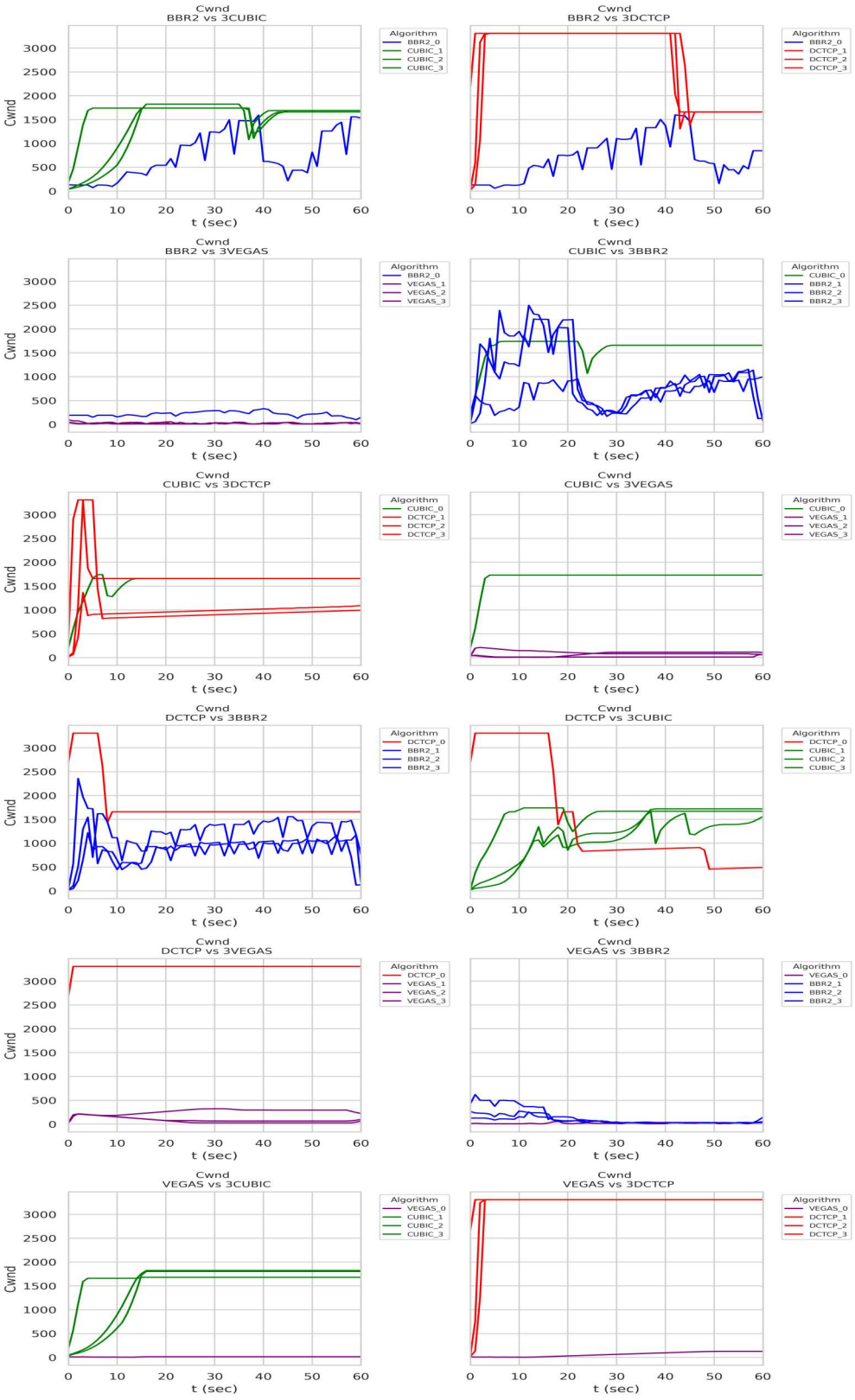


Figure 5.30: CWND for algorithms in Asymmetric-Algorithm Competition scenario.

5.8 Chapter Conclusion

The experimental analysis of TCP congestion control algorithms within virtualized environments has provided valuable insights into the behavior and performance of various algorithms under different network conditions. Across multiple experimental scenarios, including baseline tests, delayed flow analysis, and packet loss assessments, it became clear that the choice of congestion control algorithm significantly impacts network performance, particularly in terms of throughput, latency, and fairness.

Algorithms like **BBR2** and **Vegas** demonstrated a tendency towards maintaining lower latency and fairer bandwidth distribution, making them suitable for environments where minimizing delay and ensuring equitable resource allocation are priorities. **BBR2**, in particular, showed robust performance across scenarios with varying latencies and packet loss, highlighting its adaptability and potential for use in dynamic virtualized environments.

On the other hand, more aggressive algorithms such as **CUBIC** and **DCTCP** excelled in maximizing throughput, particularly in scenarios where high data transfer rates are crucial. However, this often came at the cost of increased latency and potential unfairness in bandwidth distribution, which could be detrimental in environments with prevalent latency-sensitive applications.

The comparative analysis of these algorithms underscores the importance of context-specific selection when deploying TCP congestion control strategies in virtualized data centers. For instance, **BBR2** and **Vegas** are better suited for scenarios requiring stable performance under varying network conditions. Meanwhile, **CUBIC** and **DCTCP** are preferred where throughput maximization is the primary goal.

The results presented in this chapter provide a basis for network engineers and researchers to make informed decisions about deploying TCP congestion control algorithms. By understanding the strengths and limitations of each algorithm under various conditions, stakeholders can better optimize network performance to meet the specific needs of their applications and infrastructure.

CHAPTER 6

Discussion

6.1 Overview of Key Findings

This comprehensive analysis of TCP congestion control algorithms within virtualized environments unveils significant insights into the performance dynamics of Vegas, CUBIC, BBR2, and DCTCP under various operational scenarios. Our experiments highlight the distinct behaviors of these algorithms in baseline evaluations, their resilience and adaptability to basic network failures, and their competitive interactions in scenarios involving multiple algorithms. Key findings demonstrate the nuanced trade-offs between efficiency, fairness, and latency management across different congestion control strategies. For instance, Vegas's conservative approach is advantageous in latency-sensitive environments, whereas CUBIC and DCTCP exhibit strengths in scenarios prioritizing throughput. BBR2, with its balanced approach, stands out for its scalability and fairness, particularly in densely populated virtual environments, showcasing its potential as a versatile solution in cloud data centers.

The implications of these findings extend to the design and management of virtualized network architectures, underscoring the importance of selecting and tuning congestion control algorithms to suit specific operational needs and environmental conditions. The adaptability of BBR2 in maintaining performance stability across varying network densities, its resilience to packet loss, and its capability to ensure fairness in competitive settings illustrate the algorithm's suitability for dynamic cloud-based infrastructures. These insights contribute to a deeper understanding of congestion control dynamics in virtualized settings and guide network administrators and system architects in optimizing network performance and reliability. The strategic selection of TCP algorithms, based on comprehensive analysis and tailored to the unique demands of virtualized environments, emerges as crucial in enhancing the efficiency and robustness of cloud data centers.

6.2 Implications of Baseline Evaluation

Vegas's lower sending rates and conservative CWND increase suggest its applicability in scenarios where preventing network congestion is paramount, advocating for its

selection in latency-sensitive environments. Conversely, the higher sending rates and aggressive CWND increase demonstrated by **CUBIC** and **DCTCP** align with use cases that prioritize maximizing data transfer rates, even at the cost of higher latency. **BBR2**'s performance, characterized by a balance between throughput efficiency and fairness, indicates its potential as a scalable solution for complex data center networks, where equitable resource distribution among a high number of VMs is critical.

The observation of retransmissions, particularly pronounced in **BBR2** during the initial flow stages, underscores the algorithms' adaptability to network conditions. This adaptability is essential for maintaining stable performance across varying network densities and conditions, suggesting a need for algorithms that can quickly adjust to changing network environments.

These findings have strategic implications for network architecture design and management, especially in selecting and tuning congestion control algorithms to match specific operational requirements. For network administrators and system architects, understanding the balance between efficiency, fairness, and latency facilitated by each algorithm is crucial in optimizing network performance and reliability in cloud data centers.

The baseline evaluation of TCP congestion control algorithms—Vegas, **CUBIC**, **BBR2**, and **DCTCP**—within a virtualized environment reveals nuanced implications for their deployment in cloud data centers. In virtualized scenarios, where server-to-server communication involves mediation through multiple virtual machines (VMs) on high-performance servers, network management and configuration rely heavily on the observed behaviors of these algorithms. For instance, the conservative approach of Vegas, with its lower sending rates and RTTs, aligns with environments prioritizing latency sensitivity over throughput maximization, making it suitable for applications requiring quick response times. Conversely, **CUBIC** and **DCTCP**'s aggressive bandwidth utilization strategies, reflected in their higher sending rates and CWND values, suggest their applicability in scenarios where maximizing data transfer rates is crucial, albeit at the potential cost of increased latency and retransmissions. **BBR2**'s adaptability, demonstrated through its adjustment period and subsequent performance stability, offers a balanced approach for managing throughput efficiency and fairness in densely populated virtual environments. This adaptability is particularly relevant in cloud infrastructures, where dynamic resource allocation and fluctuating network conditions demand algorithms that can swiftly adjust to maintain optimal performance.

6.3 Analysis of Basic Network Failures

This section delves into the results of TCP congestion control algorithms under varying network delays in virtualized environments.

6.3.1 Impact of Delays in Different Scenarios

The “Impact of Delays in Different Scenarios" analysis across both "Two-Level Delayed Flow Analysis" and "Four-Level Delayed Flow Analysis" experiments offers insights into the performance and fairness of TCP congestion control algorithms under varied delay conditions within virtualized environments. The experiments reveal that BBR2 consistently maintains a higher degree of fairness and throughput efficiency, regardless of the delay, showcasing its robustness and reliability in handling latency variations without necessitating retransmissions. This suggests BBR2's potential as a preferable algorithm in cloud data center networks, where latency inconsistencies are typical, due to its ability to balance congestion window sizes adaptively and minimize latency impacts on throughput.

Conversely, while demonstrating scalability with increased flows, Vegas indicates its conservative bandwidth utilization might not always be optimal in varied-delivery scenarios. However, it improves in fairness with higher loads. CUBIC and DCTCP's performance, characterized by aggressive bandwidth claims, remains consistent across scenarios, suggesting their suitability for applications prioritizing throughput over latency. However, their larger congestion windows in no-delay scenarios highlight a potential for increased latency under competition, which could impact applications sensitive to delay.

The nuanced behaviors observed across these algorithms under delay variations underscore the critical need for algorithm selection and tuning tailored to specific network conditions and application requirements within virtualized environments. BBR2's adaptability to delay conditions without sacrificing fairness or necessitating retransmissions positions it as a compelling option for managing network traffic in cloud-based infrastructures facing dynamic latency challenges. These findings contribute to a deeper understanding of TCP congestion control dynamics in virtualized settings and guide network administrators in optimizing configurations to achieve desired performance outcomes in complex, latency-variable network ecosystems.

The virtualization perspective. In a cloud-based data center environment characterized by fluctuating network conditions due to virtualization overhead and resource multiplexing, the adaptability and fairness of algorithms like BBR2 are crucial. While BBR2's mechanism, which relies on RTT measurements to estimate available bandwidth, generally ensures robust network performance, it is susceptible to inaccuracies in

environments where virtualization layers dynamically change their load, potentially affecting RTT estimations. This can lead to misestimation of bandwidth and consequently impact performance and fairness. Recognizing this limitation, further investigation into new ways in which to adapt BBR2 could or enhanced to better handle such fluctuations in virtualized settings is essential. Nevertheless, the overall performance of BBR2 in maintaining throughput fairness with minimal latency impact highlights its suitability for cloud data centers. Ensuring consistent and predictable network performance remains critical for supporting diverse and latency-sensitive applications, and BBR2, despite its limitations, essentially meets these requirements.

6.3.2 Packet Loss Effects

The experiments investigating the impact of packet loss on TCP congestion control algorithms within virtualized environments, both on a single VM and across two VMs, reveal insights into each algorithm's resilience and adaptability to packet loss. BBR2's performance stands out in maintaining higher sending rates and throughput, even as packet loss rates increase, showcasing its robustness and superior handling of packet loss. This resilience is attributed to BBR2's ability to sustain larger congestion windows and effectively discern between congestion-induced losses and those arising from other factors, thereby minimizing the negative impact on network performance. Unlike traditional TCP congestion control algorithms that primarily rely on a single metric, such as packet loss or delay, to infer network congestion, BBR2 employs a more holistic strategy. This involves the utilization of multiple signals, including bandwidth estimation and round-trip time (RTT) measurements, to form a nuanced understanding of the network's state. By not solely depending on packet loss or delay variations, BBR2 can differentiate between congestion-induced packet loss and packet loss resulting from other causes, such as transient network errors or link failures.

Furthermore, BBR2's mechanism to maintain more oversized congestion windows under packet loss scenarios is rooted in its ability to gauge the network's available bandwidth and end-to-end delay accurately. This accurate bandwidth estimation allows BBR2 to adjust its sending rate proactively rather than reactively downsizing the congestion window after a loss is detected, a common approach in loss-based algorithms. Consequently, BBR2 can sustain higher levels of throughput, as it avoids the conservative reduction in sending rates that characteristically hampers other algorithms in the face of packet loss.

Moreover, BBR2 introduces improvements in loss detection and response, allowing it to respond to actual congestion signals with more precision and less dependency on arbitrary thresholds. This nuanced approach ensures that BBR2 can maintain operational efficiency and fairness, even in networks where packet loss does not directly signal congestion. BBR2's strategy is to utilize a comprehensive set of network performance

indicators, enabling it to sustain robust and efficient transmission rates by accurately interpreting the network's congestion state without over-relying on any single metric.

Conversely, despite struggling with lower throughput and sending rates under packet loss conditions, Vegas demonstrates improved fairness and balance in sending rates across flows in the two VM scenarios, especially as packet loss increases. This suggests that Vegas and BBR2 offer a more equitable distribution of network resources among competing flows, an essential trait for maintaining performance consistency in environments with variable network quality.

CUBIC and DCTCP, while exhibiting less sensitivity to increased packet loss in sending rate and throughput, tend to have higher congestion windows for lossless flows, potentially exacerbating unfairness in bandwidth allocation among flows with varying packet loss rates. Their higher RTTs and fewer retransmissions, compared to BBR2 and Vegas, reflect a different approach to handling packet loss, prioritizing bandwidth utilization over fairness or latency in these scenarios. However, it is crucial to note that DCTCP offers significant advantages in data center environments that other algorithms do not manage effectively. Specifically, DCTCP's utilization of Explicit Congestion Notification (ECN) allows for more granular control of congestion, which is particularly effective in addressing the incast problem common to data center traffic. This capability makes DCTCP highly suitable for environments with dense server deployments and high volumes of parallel data retrieval operations, where minimizing congestion spikes and avoiding packet drops are critical.

The findings underscore the importance of choosing TCP congestion control algorithms that perform well under ideal conditions and maintain fairness and efficiency in the face of packet loss, a common challenge in virtualized cloud environments. BBR2's adaptability and robustness under packet loss conditions and its ability to maintain low RTTs and high throughput position it as a strong candidate for scenarios where network reliability and performance consistency are paramount.

The virtualization perspective. In virtualized environments, where resources are dynamically shared among multiple virtual machines (VMs) on the same physical hardware, the impact of packet loss on TCP congestion control algorithms becomes even more significant. The experiments highlighting the effects of packet loss on single and multiple VMs demonstrate the critical importance of algorithm resilience and fairness in such settings. Specifically, BBR2's robust performance under packet loss conditions—maintaining high throughput and low RTTs aligns well with the demands of virtualized data centers, where fluctuating network quality and resource competition are commonplace. This resilience ensures that applications running on VMs can continue to communicate efficiently, even in the face of network challenges. The varying responses of TCP congestion control algorithms to packet loss, as observed in the experiments, underscore the importance of selecting and tuning these algorithms to suit the unique

characteristics of virtualized environments, where ensuring consistent and reliable network performance is crucial for supporting a wide range of applications and services.

6.4 Insights from Multiple Algorithm Competitions

The insights drawn from the "Dual-Algorithm Competition (2 VMs)" and "Asymmetric Competition (4 VMs)" scenarios within the multiple algorithms competition experiments offer a look at the dynamics of TCP congestion control algorithms when faced with direct network resource competition. In the dual-algorithm setup, the aggressive bandwidth utilization strategies of DCTCP and CUBIC allowed them to dominate over the more conservative approaches of VEGAS and BBR2, particularly evident in their ability to maintain larger congestion windows and achieve higher sending rates.

Conversely, the asymmetric competition scenario, involving one VM against three operating under a different algorithm, highlights the adaptability and resilience of BBR2 in the face of uneven competition. Despite the inherent disadvantages in bandwidth usage and sending rates, BBR2's performance against multiple instances of CUBIC and DCTCP demonstrates its capability to maintain fairness over time, a crucial attribute in diverse network conditions. CUBIC's consistent performance, maintaining higher throughput even when solo against multiple BBR2 or DCTCP flows, further exemplifies its robustness in monopolizing available bandwidth.

With its conservative nature, VEGAS consistently found itself at a disadvantage across both scenarios, struggling to compete effectively against the more aggressive algorithms. This indicates a potential mismatch between VEGAS's conservative bandwidth utilization strategy and high competition and resource contention environments.

These experiments underscore the significance of algorithm selection based on the competitive landscape of the network environment. The ability of BBR2 to adapt and compete for resources, even when outnumbered, alongside CUBIC's aggressive and dominant performance, provides critical insights for network administrators in optimizing congestion control strategies. It highlights the necessity of understanding the balance between aggression, efficiency, and fairness in managing network resources, especially in virtualized environments where the competition for bandwidth is a constant challenge.

The virtualization perspective. The aggressive behaviors of DCTCP and CUBIC, which enable these algorithms to secure more bandwidth when competing, highlight the potential for performance optimization in environments where network resources are heavily contested. However, this advantage also points to the risk of creating imbalances in resource allocation, particularly in virtualized settings where multiple applications or services may share physical resources. The adaptability and resilience of BBR2, espe-

cially in maintaining fairness in asymmetric competitions, illustrate the importance of algorithmic flexibility in environments characterized by fluctuating demand and diverse workload profiles. The challenges VEGAS faces in these competitive scenarios reveal the limitations of more conservative algorithms in aggressive competition landscapes, emphasizing the need for network administrators to carefully consider the specific characteristics and demands of their virtualized environments when selecting and tuning congestion control algorithms. This strategic decision-making is essential to ensuring that virtualized networks can efficiently support a wide range of applications while maintaining high levels of performance and fairness, even under intense competition for bandwidth.

Comparison with Existing Literature

In comparing the results obtained in this dissertation with those reported in the related works, it becomes evident how virtualized environments' specific conditions and configurations influence TCP congestion control algorithm performance differently from traditional network settings.

Turkovic et al. [TKU19] focused on a controlled test environment to analyze the performance of TCP congestion control algorithms under various network conditions. Their study revealed that loss-based algorithms are more aggressive, resulting in high retransmission rates and latency, while delay-based algorithms are more conservative, presenting, in general, lower throughput rates. Hybrid algorithms like BBR balance these extremes, but favor flows with higher RTT. In this dissertation, similar results were observed: loss-based algorithms showed aggressive behavior with many retransmissions and higher delay. In contrast, delay-based algorithms maintained low RTT but faced difficulties maintaining a high throughput rate under competition.

In their “Base-Line scenario", Turkovic et al. found that delay-based algorithms, such as Vegas, were more conservative, resulting in lower throughput. Loss-based algorithms like Cubic were aggressive and consequently had high numbers of retransmissions. Our work observed similar results in virtualized environments, adding that virtualization overhead further influenced throughput and RTT metrics. In both studies, hybrid algorithms such as BBR and BBR2 showed intermediate performance, but virtualization introduced additional variations in performance, such as latency spikes and variations in bandwidth utilization.

In the “bandwidth scenario (BW)", Turkovic et al. noted that delay-based algorithms lose performance when competing with loss-based or hybrid algorithms. They also identified that hybrid algorithms, such as BBR, showed better fairness properties than loss-based ones. Our work also observed that, in virtualized environments, delay-based algorithms were at a disadvantage in competitive scenarios and that BBRv2 showed

fairer performance in terms of resource sharing. However, the virtualization overhead has increased unpredictability, resulting in a less stable fairness. For example, multiple VMs competing for the same physical resources can lead to uneven bandwidth allocation and variability in throughput, hurting the observed inter-fairness.

Turkovic et al. concluded that delay-based algorithms had better inter-fairness properties, ensuring a fair distribution of resources between different algorithms operating on the same network, with a Jain fairness index close to 1, indicating that available resources were shared fairly between the flows. Hybrid algorithms, such as BBR, also demonstrated good fairness properties but were less consistent than delay-based ones. In virtualized environments, we observed that virtualization overhead added significant variability, negatively impacting inter-fairness. Loss-based algorithms like Cubic showed more significant throughput oscillations and lower fairness than when used in a non-virtualized environment.

Both studies observed that loss-based algorithms such as CUBIC achieve good intra-fairness properties when competing against each other, with flows eventually converging to an equitable bandwidth distribution. However, in our work, we noticed that this convergence can be slower and less predictable in virtualized environments due to the dynamics of VM resource allocation. Hybrid algorithms, such as BBRv2, have improved over the original BBR but still face fairness challenges in virtualized environments. Bandwidth distribution between flows using BBRv2 was observed to be fairer, but variability in VM resource allocation resulted in latency spikes and fluctuating throughput, especially under varying workloads.

Finally, in the "RTT scenario," Turkovic et al. found that hybrid algorithms like BBR favored flows with higher RTT. In contrast, delay-based algorithms were able to maintain low RTTs but suffered when competing with loss-based algorithms. This research observed that BBRv2 often favored flows with **lower RTT** in virtualized environments. Several factors can explain this difference in behavior, including the influence of hypervisor overhead, the jitter introduced by virtualization, and variability in dynamic resource allocation and hypervisor resource scheduling policies that may favor flows with lower RTT. In virtualized environments, the hypervisor manages the distribution of physical resources among VMs, introducing variability and additional overhead. These scheduling policies can result in a resource allocation that favors flows with lower RTT, as these flows can respond more quickly to changes in resource allocations, leading to more stable and efficient performance. Furthermore, hypervisor overhead and the introduction of jitter can be more detrimental to flows with higher RTT, which are more sensitive to these variations. However, it is important to note that BBRv2 does not always favor flows with lower RTT in virtualized environments. As illustrated in our study, different behaviors were observed depending on the number of flows. For example, in a scenario with four flows, BBRv2 allowed flows with a delay of 200ms to access bandwidth greater than or equal to flows with 0ms delay over time. This contrasts with

the scenario with only two flows, where BBRv2 prioritized the flow with lower RTT. This behavior is further discussed in Section 5.2. In summary, virtualization introduces an additional layer of complexity, with dynamic resource allocation impacting RTT metrics more significantly than in Turkovic et al.'s controlled tests. However, the dynamic adjustments made by BBRv2 help optimize throughput and minimize latency across various network conditions.

Therefore, while the fundamental behavior of TCP congestion control algorithms is consistent between the two studies, the virtualization context in our research reveals an additional layer of challenge in deploying these algorithms in real virtualization-based networks compared to controlled test environments analyzed by Turkovic et al. This crucial difference highlights the importance of considering the effects of virtualization when evaluating the performance of TCP congestion control algorithms in modern data center scenarios.

Nguyen et al. [NGS16] used the ns-3 network simulator to evaluate TCP algorithms in Data Center Networks (DCNs), finding significant performance variances among algorithms like Vegas, which kept queue lengths low, and BIC and CUBIC, which struggled with high packet drop rates and queue occupancy. Our physical experiments highlight similar trends in algorithm performance concerning queue management and packet drops. However, introducing virtualized components adds complexity, with hypervisor overhead and virtual network functions influencing algorithm behavior more significantly than non-virtualized DCNs. For instance, our results show that CUBIC's aggressiveness, beneficial in traditional settings for maximizing throughput, can lead to inefficient bandwidth utilization and increased latency in virtualized environments due to the additional processing required by virtual switches and hypervisors. It's critical to note that CUBIC does not directly adapt to such additional processing overhead. Instead, the increased likelihood of packet loss and variable delay imposed in these environments might cause CUBIC to incorrectly perceive network conditions, leading to inefficient bandwidth utilization and potentially increased latency.

Abadleh et al.'s [ATB+22] comparative analysis indicates that TCP Vegas had the best performance in terms of throughput, latency, and retransmission rate due to packet loss in the scenario with a single TCP flow under congestion when compared to TCP Tahoe, Reno, and New Reno. However, in the scenario with two TCP flows and one background UDP flow, Vegas' performance in terms of throughput was lower than that of TCP Tahoe, a loss-based algorithm, due to reduction of its aggressive in the sending rate when detecting congestion, allowing Tahoe to utilize more bandwidth. In our experiments, TCP Vegas also showed good performance in terms of latency and retransmission rate when in a scenario where it was present on its own. However, Vegas' performance in terms of throughput was lower than that of the other compared algorithms, including Cubic, which is loss-based like Tahoe, due to its aggressive reduction in the sending rate when detecting congestion, and this difference was more accentuated

due to virtualization-related issues, such as hypervisor overhead and variability in VM resource allocation, which can affect the efficiency of Vegas congestion control.

Patel et al. [PSK+20] reported that CUBIC and BIC were more effective in environments with high bandwidth and latency, attributing this to their aggressive congestion window growth strategies. However, CUBIC's performance demonstrated a nuanced complexity in our virtualized context. While its aggressive window growth strategy aimed to maximize throughput, we observed that this approach can lead to suboptimal performance in virtualized environments in some cases. The virtualization layer introduces additional latency and packet processing overhead, which can mitigate the advantages of CUBIC's aggressive window growth. For example, in scenarios with high VM density, CUBIC's rapid increase in sending rate often resulted in increased packet queuing at the virtual switch, leading to higher latency and occasionally greater packet loss when the queues overflowed. This highlights the critical impact of virtualization-specific factors on algorithm performance. Our results suggest that while CUBIC remains a robust choice for traditional networks, its efficacy in virtualized environments may be compromised, necessitating adjustments or the consideration of alternative algorithms better suited to the unique challenges presented by virtualization.

In summary, this dissertation's results extend the understanding of TCP congestion control algorithm performance by highlighting the distinct impact of virtualized environments. Our study underscores the need for further optimization. The nuanced behaviors observed in our experiments, particularly regarding the interaction between algorithmic strategies and virtualization-specific factors, point to the complexity of ensuring optimal performance in such environments and suggest avenues for future research and development.

Practical Implications

The experimental analysis of TCP congestion control algorithms—Vegas, CUBIC, BBR2, and DCTCP—within virtualized environments offers profound practical implications for network management, particularly in cloud data centers. These findings guide network administrators and system architects in selecting and fine-tuning congestion control algorithms to optimize network performance, reliability, and fairness in varying operational scenarios. Each algorithm exhibits unique characteristics that align with specific network management goals, from latency sensitivity to throughput maximization and fairness in resource allocation. Based on our experimental results, Table 6.1 provides recommendations targeted towards each congestion control algorithm under different network scenarios, guiding network administrators in optimizing their network configurations.

For latency-sensitive applications, such as real-time communication and interactive

Table 6.1: Recommendation Table. Cells with two values show that both algorithms were equally relevant. The "-" symbol indicates that no algorithm was better for that metric or that all algorithms performed poorly.

Scenario	**Metrics**			
	Most Data Successfully Delivered	**Best RTT**	**More Fairness**	**Less Retransmission**
Baseline	● CUBIC / ● DCTCP	● VEGAS	● BBR2	● VEGAS
Basic Failures (Delay)	● BBR2	-	● BBR2 / ● CUBIC	● BBR2
Basic Failures (Drop)	● BBR2	● VEGAS	● BBR2 / ● VEGAS	-
Multiple	● CUBIC / ● DCTCP	● VEGAS	● BBR2	● VEGAS

services, Vegas's conservative congestion control mechanism can prevent network congestion effectively, maintaining lower round-trip times (RTTs) and reducing the likelihood of packet retransmissions. This approach ensures quick response times, which is crucial for enhancing user experience in latency-critical environments. As shown in Table 6.1, Vegas consistently provided the best RTT in the Baseline, Basic Failures, and Multiple scenarios, making it an excellent choice for applications where low latency is essential. On the other hand, scenarios that prioritize data transfer rates over latency, such as bulk data transfers and large-scale backups, may benefit from the aggressive bandwidth utilization strategies of CUBIC and DCTCP. These algorithms are designed to maximize throughput, albeit potentially at the cost of increased latency and higher retransmission rates, which may be acceptable trade-offs in contexts where data delivery speed is paramount.

BBR2's balanced performance profile, characterized by its adaptability and efficiency in maintaining throughput fairness and minimizing RTTs, makes it particularly suitable for complex, densely populated data center networks. Its ability to quickly adjust to changing network conditions without necessitating frequent retransmissions positions BBR2 as a scalable solution for cloud data centers, where equitable resource distribution among a high number of virtual machines (VMs) is critical. As indicated in Table 6.1, BBR2 excelled in fairness across various scenarios. It demonstrated resilience with regard to delay and packet drop situations, underscoring its potential as a reliable congestion control option in environments where network reliability and performance consistency are paramount.

The practical implications of these insights extend to the strategic design and management of virtualized network architectures. Understanding the specific strengths and

limitations of each TCP congestion control algorithm allows network administrators to tailor their approach to meet the diverse requirements of different applications and services hosted in cloud data centers. Additionally, the dynamic nature of cloud-based infrastructures, characterized by fluctuating network conditions and diverse workload profiles, demands a flexible approach to network management. The findings from the analysis of basic network failures and multiple algorithm competitions highlight the importance of algorithm adaptability in maintaining stable and fair network performance across varying scenarios. Continuous monitoring, analysis, and adjustment of congestion control strategies are essential to ensuring that virtualized networks can efficiently support a wide range of applications while adapting to the evolving demands of cloud data centers.

Key Takeaways

Based on the results summarized in Table 6.1, we provide the following key takeaways to guide network administrators in selecting the most appropriate congestion control algorithm for virtualized environments. These recommendations apply to scenarios analogous to this study's investigation.

- **Latency-Sensitive Applications:** For a virtualized environment hosting latency-sensitive applications, such as real-time communication and interactive services, Vegas is suggested. It consistently provided the best RTT in the Baseline, Basic Failures (Drop), and Multiple scenarios.
- **High-Throughput Requirements:** In virtualized scenarios where maximizing data transfer rates is crucial, such as bulk data transfers and large-scale backups, CUBIC and DCTCP tend to be favorable choices. These algorithms excelled with regard to metrics representing exchanged data for the Baseline and Multiple scenarios.
- **Fairness and Resilience:** BBR2 tends to be well-suited for virtualized environments requiring balanced performance and fairness. It demonstrated superior fairness across various scenarios and showed resilience to delay and packet drop situations, making it suitable for complex, densely populated data center networks.

General Recommendation: Network administrators managing virtualized environments should select TCP congestion control algorithms based on specific operational needs. Vegas is ideal for minimizing latency, CUBIC and DCTCP for maximizing throughput, and BBR2 for maintaining fairness and adaptability in dynamic network conditions. As mentioned, these recommendations are particularly relevant for scenarios analogous to this study's investigation.

CHAPTER 7

Conclusion

7.1 Limitations and Future Research

Limitations

The experimental analysis conducted in this work focuses on TCP congestion control algorithms in virtualized environments and provides insights into their dynamics and performance in the evaluated context. This study uses a specific virtualization platform, KVM, and selected four TCP algorithms (Vegas, CUBIC, BBR2, DCTCP) due to time and resource constraints. While limiting the generalization of findings across different platforms or algorithms, these choices allow for in-depth exploration within a manageable scope. Despite the diversity in simulated network conditions, they do not encompass the total variability of real-world environments. This limitation is a trade-off needed for maintaining controlled and reproducible experimental conditions. Furthermore, the scale of the experiments was determined subject to resource availability, which inherently limits the exploration of interactions between multiple algorithms under competitive conditions. The observed performance of TCP algorithms is also linked to their specific implementations in the Linux kernel, which may change over time.

Furthermore, this study did not include the QUIC protocol in its experiments, focusing only on TCP. The decision to exclude QUIC was due to several factors. First, QUIC operates over UDP, introducing a different set of implementation dynamics and complexities compared to TCP [PTFK23]. Integrating and testing QUIC would require significant modifications to the experimental setup, including changes to traffic generation tools and monitoring frameworks. Additionally, QUIC is still evolving, and its various versions may not be as stable or widely supported in virtualized environments as TCP. Second, although QUIC is gaining popularity, according to recent data from Cloudflare Radar [Clo24] on adopting and using transport protocols on the internet, TCP demonstrates significant dominance over QUIC. The Cloudflare Radar reveals that, from July 2023 to July 2024, a combination of the Transport Layer Security (TLS) 1.2 and TLS 1.3 protocols, both running over TCP, made up approximately 69% of secure traffic. In contrast, QUIC, which incorporates TLS 1.3 for security, represents 30.5% of secure traffic. Given the focus on TCP and time and resource constraints, the inclusion of QUIC was beyond the feasible scope of this study. However, the performance charac-

teristics of QUIC in cloud and data center environments would be an interesting avenue for future research.

Future Research

Several promising areas can be explored to build upon the findings of this dissertation. Future research could explore TCP congestion control across different virtualization technologies, such as Docker, VMware, LDoms/Oracle VM Server, or XEN, to assess whether the results from KVM remain consistent across platforms. Including newer or alternative TCP algorithms might provide a deeper understanding of congestion control strategies and their applications. Real-world deployment with cloud service providers could validate laboratory results under operational network conditions, offering insights into the practical effectiveness of different congestion control mechanisms. Investigating the potential of machine learning and AI to predict and adjust TCP performance in real time could lead to more adaptive congestion control strategies. Additionally, the impact of hardware innovations like Software-Defined Networking (SDN) and Network Function Virtualization (NFV) on TCP congestion control deserves further exploration to optimize network traffic management in modern network infrastructures. Exploring how these technologies mitigate data center network issues like TCP incast, microbursts, and congestion spreading is also essential. These phenomena, particularly in high-throughput, low-latency environments, can severely disrupt network performance and reliability. Future investigations can enhance network management solutions' robustness, adaptability, and efficiency by extending research to these areas, particularly in virtualized environments.

7.2 Conclusion

This dissertation was dedicated to carrying out exploratory research in the domain of TCP congestion control algorithms in virtualized environments. Through a methodical experimental structure, this study observed the complex interaction between mechanisms of TCP congestion control algorithms – specifically Vegas, CUBIC, BBR2, and DCTCP – and the operational scenarios of virtualized infrastructures. The findings provide evidence of the distinctive performance characteristics of these algorithms, highlighting the critical role of strategic selection and optimization in increasing network efficiency, fairness, and reliability in cloud-based systems.

Experiments were carried out to dissect the operational dynamics of each algorithm under a spectrum of conditions, starting with baseline network operations going through simulated network failures and competitive scenarios involving multiple algorithms. The experimental analysis produced critical insights, particularly the differential adaptability of these algorithms to the challenges and opportunities presented

by virtualized environments. For example, the latency-sensitive nature of Vegas, the throughput-oriented strategies of CUBIC and DCTCP, and BBR2's balanced approach between maximizing bandwidth and minimizing latency show the diverse approaches available to network administrators to adapt network performance to specific application needs and operating conditions.

The implications of these findings provide tangible, practical guidance for designing and managing virtualized networks. This study highlights the importance of a nuanced understanding of TCP congestion control algorithms to optimize the performance of cloud data centers. It highlights the need for network administrators and systems designers to adopt a dynamic and informed approach to algorithm selection, ensuring that virtualized networks can meet the growing demands of cloud computing with agility and efficiency.

Furthermore, this research contributes to filling the significant gap in the literature on TCP congestion control in virtualized environments. By providing empirical evidence and differentiated analysis of the performance of these algorithms in such environments using a physical testbed, the study improves our understanding of the dynamics of virtualized networks. It provides a foundation for future research, encouraging further investigation into the complex interactions between network protocols and virtualization technology.

In conclusion, this document answers the research questions posed at the outset and opens new avenues for exploration and application. A continuous evolution of network management practices in cloud data centers is required, driven by an increasingly more profound understanding of the interaction between TCP congestion control algorithms and virtualized environments. Congestion control significantly impacts the design and optimization of virtualized networks, ensuring that they are robust, efficient, and capable of supporting the growing demands of the digital era.

Bibliography

[AB21] Ahmed M Abdelmoniem and Brahim Bensaou. Implementation and evaluation of data center congestion controller with switch assistance. *arXiv preprint arXiv:2106.14100*, 2021.

[ACA21] Abhineet Anand, Amit Chaudhary, and M Arvindhan. The need for virtualization: when and why virtualization took over physical servers. In *Advances in Communication and Computational Technology: Select Proceedings of ICACCT 2019*, pages 1351–1359. Springer, 2021.

[AGM+10] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74, 2010.

[APR15] Md Imran Alam, Manjusha Pandey, and Siddharth S Rautaray. A comprehensive survey on cloud computing. *International Journal of Information Technology and Computer Science (IJITCS)*, 7(2):68, 2015.

[APS99] M. Allman, V. Paxson, and W. Stevens. RFC2581—TCP congestion control. Technical report, RFC, 1999.

[ASABB19] Rasool Al-Saadi, Grenville Armitage, Jason But, and Philip Branch. A survey of delay-based and hybrid tcp congestion control algorithms. *IEEE Communications Surveys & Tutorials*, 21(4):3609–3638, 2019.

[ATB+22] Ahmad Abadleh, Aya Tareef, Alaa Btoush, Alaa Mahadeen, Maram M Al-Mjali, Saqer S Alja'Afreh, and Anas Ali Alkasasbeh. Comparative analysis of tcp congestion control methods. In *2022 13th International Conference on Information and Communication Systems (ICICS)*, pages 474–478. IEEE, 2022.

[ATRK10] Alexander Afanasyev, Neil Tilley, Peter Reiher, and Leonard Kleinrock. Host-to-host congestion control for tcp. *IEEE Communications Surveys I& Tutorials*, 12(3):304–342, 2010.

[Bad] Mukhtiar Badshah. What is a virtual data center? - open source listing. https://www.opensourcelisting.com/what-is-a-virtual-data-center (Accessed on 2024-02-01).

[Bel13] Anton Beloglazov. Energy-efficient management of virtual machines in data centers for cloud computing. 2013.

[BP95] Lawrence S. Brakmo and Larry L. Peterson. Tcp vegas: End-to-end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, 1995.

[CCG+17] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Communications of the ACM*, 60(2):58–66, 2017.

[CCY+19] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Ian Swett, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, Matt Mathis, and Van Jacobson. BBR v2: A Model-based Congestion Control [Slides]. In *IETF 104: Prague, March 2019*. Retrieved from https://datatracker.ietf.org/meeting/104/materials/slides-104-iccrg-an-update-on-bbr-00, 2019.

[CFH+05] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286, 2005.

[Clo24] Cloudflare. Adoption and usage, 2024. Accessed: 2024-07-01.

[CRBV+16] Bryce Cronkite-Ratcliff, Aran Bergman, Shay Vargaftik, Madhusudhan Ravi, Nick McKeown, Ittai Abraham, and Isaac Keslassy. Virtualized congestion control. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 230–243, 2016.

[dFdC+23] Assis T. de Oliveira Filho, Eduardo Freitas, Pedro R.X. do Carmo, Djamel F.H. Sadok, and Judith Kelner. Measuring the impact of sr-iov and virtualization on packet round-trip time. *Computer Communications*, 211:193–215, 2023.

[DLZ+15] 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.

[dOFFdC+22] Assis T de Oliveira Filho, Eduardo Freitas, Pedro RX do Carmo, Djamel HJ Sadok, and Judith Kelner. An experimental investigation of round-trip time and virtualization. *Computer Communications*, 184:73–85, 2022.

[DSFJ15] Ramide Dantas, Djamel Sadok, Christofer Flinta, and Andreas Johnsson. Kvm virtualization impact on active round-trip time measurements. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 810–813, 2015.

[FH99] S. Floyd and T. Henderson. RFC2582—the NewReno modification to TCP's fast recovery algorithm. Technical report, RFC, 1999.

[FL03] Cheng Peng Fu and Soung C Liew. Tcp veno: Tcp enhancement for transmission over wireless access networks. *IEEE Journal on selected areas in communications*, 21(2):216–228, 2003.

[FRE21] Eduardo Felipe Fonseca de FREITAS. Experimental evaluation on packet processing frameworks under virtual environments. Master's thesis, Universidade Federal de Pernambuco, 2021.

[Got11] Yasunori Goto. Kernel-based virtual machine technology. *Fujitsu Scientific and Technical Journal*, 47(3):362–368, 2011.

[HRX08] 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.

[Jac88] Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM computer communication review*, 18(4):314–329, 1988.

[Jac90] V. Jacobson. Modified TCP congestion avoidance algorithm. email to the end2end list, April 1990.

[Jai84] Raj Jain. A quantitative measure of fairness and discrimination for resource allocation in shared systems. *DEC, Sep. 1984*, 1984.

[JWL+05] Cheng Jin, David Wei, Steven H Low, Julian Bunn, Hyojeong D Choe, John C Doylle, Harvey Newman, Sylvain Ravot, Suresh Singh, Fernando Paganini, et al. Fast tcp: From theory to experiments. *IEEE network*, 19(1):4–11, 2005.

[KGCBH20] Elie F. Kfoury, Jose Gomez, Jorge Crichigno, and Elias Bou-Harb. An emulation-based evaluation of tcp bbrv2 alpha for wired broadband. *Computer Communications*, 161:212–224, 2020.

[KKL+07] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux symposium*, volume 1, pages 225–230. Dttawa, Dntorio, Canada, 2007.

[LBS06] Shao Liu, Tamer Başar, and Ravi Srikant. Tcp-illinois: A loss and delay-based congestion control algorithm for high-speed networks. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, pages 55–es, 2006.

[LZ15] Yifei Lu and Shuhong Zhu. Sdn-based tcp congestion control in data center networks. In *2015 IEEE 34th international performance computing and communications conference (IPCCC)*, pages 1–7. IEEE, 2015.

[MLD+15] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review*, 45(4):537–550, 2015.

[NG23] C Nandhini and Govind P Gupta. Exploration and evaluation of congestion control algorithms for data center networks. *SN Computer Science*, 4(5):509, 2023.

[NGS16] Truc Anh N. Nguyen, Siddharth Gangadhar, and James P. G. Sterbenz. Performance evaluation of tcp congestion control algorithms in data center networks. In *Proceedings of the 11th International Conference on Future Internet Technologies*, CFI '16, page 21–28, New York, NY, USA, 2016. Association for Computing Machinery.

[Pos81] Jon Postel. Rfc0793: Transmission control protocol, 1981.

[PSK+20] Sanjeev Patel, Yash Shukla, Nikhil Kumar, Tejasv Sharma, and Kulgaurav Singh. A comparative performance analysis of tcp congestion control algorithms: Newreno, westwood, veno, bic, and cubic. In *2020 6th International Conference on Signal Processing and Communication (ICSC)*, pages 23–28. IEEE, 2020.

[PTFK23] Haorui Peng, William Tarneberg, Emma Fitzgerald, and M. Kihl. Performance evaluation of quic vs. tcp for cloud control systems. *2023*

International Conference on Software, Telecommunications and Computer Networks (SoftCOM), pages 1–6, 2023.

[RAZ20] Amarildo Rista, Jaumin Ajdari, and Xhemal Zenuni. Cloud computing virtualization: A comprehensive survey. In *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 462–472. IEEE, 2020.

[Tec23] Park Place Technologies. Traditional data center vs virtualization: Differences, 2023. Accessed: 2024-06-22.

[TKU19] Belma Turkovic, Fernando A Kuipers, and Steve Uhlig. Fifty shades of congestion control: A performance and interactions evaluation. *arXiv preprint arXiv:1903.03852*, 2019.

[TSZS06] Kun Tan, Jingmin Song, Qian Zhang, and Murad Sridharan. A compound tcp approach for high-speed and long distance networks. In *Proceedings-IEEE INFOCOM*, 2006.

[TTS15] Rohit P Tahiliani, Mohit P Tahiliani, and K Chandra Sekaran. Tcp congestion control in data center networks. In *Handbook on Data Centers*, pages 485–505. Springer, 2015.

[XHR04] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *IEEE INFOCOM 2004*, volume 4, pages 2514–2524. IEEE, 2004.

[XZLC20] Chi Xu, Jia Zhao, Jiangchuan Liu, and Fei Chen. Revisiting multipath congestion control for virtualized cloud environments. *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, pages 1–10, 2020.

This volume has been typeset in LaTeXwith the UFPEThesis class (www.cin.ufpe.br/~paguso/ufpethesis).