STABLE AND SCALABLE CONGESTION CONTROL FOR HIGH-SPEED HETEROGENEOUS NETWORKS

A Dissertation

by

YUEPING ZHANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2008

Major Subject: Computer Engineering

STABLE AND SCALABLE CONGESTION CONTROL FOR HIGH-SPEED HETEROGENEOUS NETWORKS

A Dissertation

by

YUEPING ZHANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Dmitri Loguinov
Committee Members,	Riccardo Bettati
	Jianer Chen
	A. L. Narasimha Reddy
Head of Department,	Valerie E. Taylor

May 2008

Major Subject: Computer Engineering

ABSTRACT

Stable and Scalable Congestion Control for

High-Speed Heterogeneous Networks. (May 2008)

Yueping Zhang, B.S., Beijing University of Aeronautics and Astronautics

Chair of Advisory Committee: Dr. Dmitri Loguinov

For any congestion control mechanisms, the most fundamental design objectives are stability and scalability. However, achieving both properties are very challenging in such a heterogeneous environment as the Internet. From the end-users' perspective, heterogeneity is due to the fact that different flows have different routing paths and therefore different communication delays, which can significantly affect stability of the entire system. In this work, we successfully address this problem by first proving a sufficient and necessary condition for a system to be stable under arbitrary delay. Utilizing this result, we design a series of practical congestion control protocols (MKC and JetMax) that achieve stability regardless of delay as well as many additional appealing properties. From the routers' perspective, the system is heterogeneous because the incoming traffic is a mixture of short- and long-lived, TCP and non-TCP flows. This imposes a severe challenge on traditional buffer sizing mechanisms, which are derived using the simplistic model of a single or multiple synchronized long-lived TCP flows. To overcome this problem, we take a control-theoretic approach and design a new intelligent buffer sizing scheme called Adaptive Buffer Sizing (ABS), which based on the current incoming traffic, dynamically sets the optimal buffer size under the target performance constraints. Our extensive simulation results demonstrate that ABS exhibits quick responses to changes of traffic load, scalability to a large number of incoming flows, and robustness to generic Internet traffic.

To my family

ACKNOWLEDGMENTS

As the long journey towards my Ph.D. comes to an end, it is the time for me to express my deep and sincere gratitude to everyone who gave me the possibility to complete this dissertation. As always, I would like to devote my utmost gratefulness to the incomparable Professor Dmitri Loguinov, who has been such a strong and supportive advisor and turned me from a novice into a mature researcher in the last five years. His pervasive commitment to research, strong demand for excellence, and relentless attention to detail will be guiding me for the rest of my research career. I am also truly grateful for the infinite patience and countless hours he spent on helping me improve my research, writing, and presentation skills.

I also want to thank Professor Jianer Chen for introducing me to research and teaching me critical thinking during my early years in the graduate school. He is like a father, mentor, and friend to me and has always been a reliable source of encouragement, support, and guidance. I am also indebted to Professors A. L. Narasimha Reddy and Riccardo Bettati for kindly serving on my advisory committee. Particularly, I greatly appreciate the opportunity provided by Dr. Reddy to participate in the PERT paper and his insightful comments on improving my work.

My thanks also go to my friends and fellow students at Texas A&M for making my seven-year stay in College Station an exciting and unforgettable experience. I extend my appreciation especially to Xueqing Yu, Hao Xiong, Nan Zhang, and Yingwei Yu for treating me like a brother and always providing me assistance. I also feel extremely fortunate to have done my doctoral studies in such a pleasant and stimulating environment as IRL with Xiaoming, Zhongmei, Derek, Seong, Min, Hsin-Tsang, and all other wonderful friends and colleagues.

During this work, I have received enormous help from anonymous reviewers of

ACM SIGCOMM, IEEE INFOCOM, IEEE/ACM Transactions on Networking, IEEE Transactions on Automatic Control, and Elsevier Computer Network Journal. I extend my warm thanks to them for assisting me to improve earlier versions of this work.

Last, but not least, I gratefully acknowledge my family – from my father who is the person I always look up to, to my mother who provides me infinite passion and energy, to my brother who is a trusted source of support that I can rely on, and to Lisha, my loving wife, who is the brightest sunshine in my life and constantly keeps me moving forward. This dissertation is the result of my five years of work accompanied and supported by you with an incredible amount of patience and confidence in me. Without you, this work would simply be impossible to complete. I love you all! I love you Lisha! To you I dedicate this dissertation!

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

1 Overview

During the last decade, the Internet has witnessed an explosive growth in the demand of network capabilities from end-applications, such as large-scale distributed science computations and experiments that operate at 100 tera FLOPS (floating-point operations per second) and require massive (hundreds of peta bytes) data transfers across the country and around the world. However, it becomes widely recognized that the current Internet may not meet future requirements or scale to ultra high-speed networks. Thus, a significant focused research effort is currently under way to design new network architectures for coherent global data networks that overcome existing difficulties and ensure a robust future Internet. *Congestion control*, as the key component of the Internet infrastructure, lies right in this territory.

The most fundamental properties required by any congestion control protocols are stability and scalability. Specifically, stability determines a system's ability to avoid oscillations in the steady-state and properly respond to external perturbations caused by the arrival/departure of flows, variation in feedback, and other transient effects. Scalability refers to capability of a system to achieve its ideal performance regardless of the system's size, which can be represented in terms of the user's population or link's capacity. Clearly, these two properties are especially important in such a large-scale, complex dynamical system as the Internet.

The journal model is *IEEE/ACM Transactions on Networking.*

However, both properties are seriously challenged by heterogeneity of the Internet, which, as illustrated in Figure 1, can be viewed from two perspectives. From the perspective of end-users, heterogeneity arises from the fact that different flow has a different routing path and therefore experiences a different round-trip delay. As demonstrated in Chapter III, heterogeneous feedback delays significantly affect the performance of a network system in terms of asymptotic stability. On the other hand, heterogeneity from the router's perspective stems from the fact that the incoming traffic at a given router is not homogeneous, but composed of a mixture of traffic with different packet sizes, transfer lengths, and underlying congestion control protocols. Thus, traditional network technologies, such as buffer sizing mechanisms, that are built upon the assumption of homogeneous incoming traffic, are no longer suitable for the current reality and may inversely impact performance of the Internet. As seen from Figure 1, this work seeks to tackle heterogeneity from both the end-user's and router's perspectives. Specifically, the goal is to design practical Internet congestion control protocols that are stable under heterogeneous feedback delays and scalable to future ultra high-speed networks, and to construct an intelligent buffer sizing scheme that are robust to realistic Internet traffic (including mixtures of short- and long-lived, TCP and non-TCP traffic) and scalable to a large number of incoming flows. I next provide overviews of these two parts in the following two sections, respectively.

2 Delay-Independent Stable Congestion Control

Since its introduction by Jacobson [47] in 1988, Internet congestion control has evolved from binary-feedback methods of AIMD/TCP [19, 111] to the more exciting developments based on optimization theory [73, 75], game theory [52, 64], and control theory

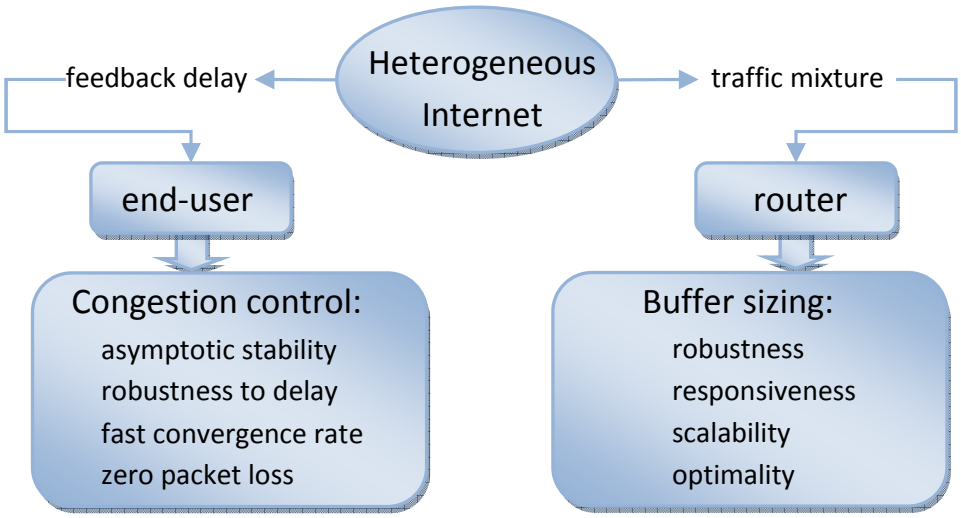


Fig. 1. Organization of the dissertation.

[51, 52, 77, 82]. It is widely recognized that TCP's congestion control in its current shape is inadequate for very high-speed networks and fluctuation-sensitive real-time multimedia applications. Thus, a significant research effort is currently under way (e.g., [30, 31, 49, 56, 60, 61, 64, 99]) to better understand the desirable properties of congestion control and develop new algorithms that can be deployed in future high-speed heterogeneous networks.

The stability issue of TCP arises as a result of the oscillatory nature of TCP's congestion control algorithm. Specifically, a TCP end-user linearly increases its sending rate in the absence of packet loss and multiplicatively decreases it upon detection of a lost packet. The binary rate-adjustment behavior causes the transmission rates of end-users to oscillate around, instead of converging to, a certain value, which is very undesirable for many Internet applications that require smooth transfer rate, such as video streaming and VoIP applications. We address this problem by utilizing a control-theoretic approach. Specifically, we follow Kelly's framework [60] and model the congestion control system as a closed-loop feedback control system. One challenge

for studying stability of such a control system is time delay, which is composed of a packet's propagation time over links and queuing time inside router buffers. The problem is further complicated by the case of *heterogeneous* delays where each user i receives its network feedback delayed by a random amount of time D_i. To better understand the relationship of delay and stability, we first conduct a comprehensive study of the effect of delay on the performance of existing Internet congestion control protocols (including TCP, rate-based AIMD, Scalable TCP, and TFRC). From this study, we demonstrate that all of the existing protocols are unable to achieve stability and exhibit undesirable behavior under non-negligible delay. Driven by this motivation, we next set our goal to build practical congestion control systems that maintain stability under heterogeneously delayed feedback as well as offering other appealing properties (e.g., fair resource allocation in the steady state, low implementation overhead, fast convergence to stationarity, and low packet loss rate).

Designing stable congestion control under delay is an active research area and has attracted a significant amount of attention from the networking community during the last five years. However, most existing papers (e.g., [22, 51, 52, 56, 66, 63, 64, 75]) model all users with homogeneous delay $D_i = D$ and do not take into account the fact that end-users in real networks are rarely (if ever) synchronized. Several recent studies [65, 77, 97] successfully deal with heterogeneous delays; however, they model D_i as a deterministic metric and require that end-flows (and sometimes routers) dynamically adapt their equations based on feedback delays, which leads to RTT-unfairness, increased overhead, and other side-effects (such as probabilistic stability). To overcome these limitations, we systematically approach this problem by first deriving a tight sufficient condition (whose necessity is also suggested by simulations) for any *single-link* congestion control system to be stable regardless of delay. Since this stability condition does not involve any delay, it is called a *delay-independent* stability

condition. Utilizing this result, we further identify several classes of systems that are stable under arbitrary delay as long as they are stable under immediate feedback. This way, stability analysis of a delayed system is reduced to that of its undelayed version, which has been well studied and can be easily carried out. In addition to its theoretical merit, this result also provides a guideline for developing practical congestion control schemes with delay-independent stability.

Employing this technique, we propose a congestion control framework called Max-min Kelly Control (MKC) [115], which introduces several novel modifications to the classical Kelly Control [60] such that the resulting system satisfied the our delay-independent stability condition obtained in [117]. We demonstrate that stability and fairness of MKC do not depend on any parameters of the network (such as delay, path length, or the routing matrix of end-users). We also show that with a proper choice of AQM feedback, MKC converges to efficiency exponentially fast, exhibits stability and fairness under *random* delays, and does not require routers to estimate any parameters of individual flows. However, EMKC has two key limitations - constant packet loss in the steady state and slow (linear) convergence rate to fairness. To overcome these two problems, we design another congestion control scheme called JetMax [116]. We establish both analytically and experimentally that JetMax exhibits zero packet loss in both the transient phase and steady state and converges to both efficiency and fairness in a fixed number of round-trip times (RTTs) regardless of link capacity and flow population. With parameters used in all `ns2` simulations and Linux experiments, for instance, it takes JetMax only 6 RTTs to reach the equilibrium under a bottleneck link with arbitrary capacity (e.g., 1 gb/s, 1 tb/s, or 1 googol (10^{100}) b/s). This, together with delay-independent stability and all its other appealing properties, makes JetMax an ideal congestion control protocol for future ultra high-speed heterogeneous networks. In addition to `ns2` simulations, we

conduct an extensive empirical evaluation of JetMax by implementing it as a Linux kernel module and an MS Windows Server 2003 NDIS driver. Our experimental results align very well with the theoretical analysis and confirm that JetMax admits a low-overhead implementation inside routers (three additions per packet). All of these characteristics make JetMax an ideal protocol for future high-speed heterogeneous networks and suggest that it, once deployed in the Internet, may remain in service for many years to come.

3 Robust Router Buffer Management

One of the key components of Internet routers is the I/O buffer, which is closely linked to various critical performance metrics, including packet loss rate, end-to-end delay, and utilization level. On one hand, router buffers should be large enough to accommodate transient bursts in packet arrivals and hold enough packets to maintain high link utilization. On the other hand, large buffers in turn leads to increased queuing delays and may potentially cause instability of TCP in certain scenarios [76]. Clearly, optimally determining the required buffer size is of immense importance for router manufactures when configuring their routers for the future high-speed Internet and significantly affects the ability of large Internet service providers to deliver and guarantee competitive Service Level Agreements (SLA) [94].

As today's Internet rapidly grows in scale and capacity, it becomes widely recognized that the classic bandwidth-delay-product (BDP) [96] rule for sizing router buffers is no longer suitable for the future Internet. In addition, the Internet is foreseeing a disruptive evolution driven by focused collaborative efforts such as the NSF Global Environment of Network Innovations (GENI) and Future Internet Network Design (FIDN) initiatives. This imposes significant challenges as well as great

opportunities for all most every corner of Internet technologies, including the next-generation infrastructure for router buffer management. As a consequence, there has emerged in the research community a surge of renewed interest [3, 5, 8, 25, 27, 37, 41, 42, 57, 68, 85, 87, 96, 102, 103] in the buffer-sizing problem during the last five years. However, these results present vastly different, even contradictory, views on how to optimally dimension the buffer of a router interface. In addition, all these results rely on certain assumptions of the incoming Internet traffic and may have limited applications to and exhibit undesirable behavior in other traffic models. In contrast, Kellett *et al.* [57] take a completely different approach and models the buffer-sizing problem as the Lur'e problem. Under this model, they proposed a dynamic buffer sizing algorithm called Adaptive Drop Tail (ADT). However, the control parameter K depends on the underlying Lur'e formulation and can hardly be obtained without off-line calculation. Thus, it still remains open to develop a *model-independent* buffer-sizing mechanism that is able to ideally allocate buffers under different traffic patterns.

In this work, we achieve the goal of buffer sizing by proposing a new buffer management infrastructure, where the router adapts its buffer size to the dynamically changing incoming traffic based on one or more performance constraints. We first formulate buffer sizing as the following problem. Let B be the total size of router's memory and $b_l(t)$ be the amount of buffer allocated to link l at time t. Then, the problem becomes determining the optimal buffer size for each link l under the constraint that $\sum_l b_l(t) \leq B$. We then propose that this problem can be alternatively solved by leveraging the *monotonic* relationship between buffer size b_l and various performance metrics (e.g., utilization u, loss rate p, and queuing delay q). Rigorously proving this relationship is very difficult and out of scope of this dissertation. Instead, we provide an intuitive explanation of this result using a simple yet generic congestion control

model.

Utilizing this result, we design a buffer management mechanism, called *Adaptive Buffer Sizing* (ABS), which dynamically determines the minimum buffer size satisfying the target performance constraints based on real-time traffic measurements. ABS consists of two sub-controllers ABS_u and ABS_p, each of which employs an Integral controller that adapts to dynamics of input traffic by regulating the buffer size based on the error between the measured and target values of utilization and loss rate, respectively. However, we observe that the naive Integral controller ABS_u drives buffers of non-bottleneck routers to infinity. We successfully address this problem by introduce a damping component, such that the resulting ABS_u quickly converges buffers to their equilibrium values in both bottleneck and non-bottleneck routers.

Another challenge is how to tune integral gains for optimal performance. Improper parameter settings may lead to undesirable system behavior, such as slow convergence and persistent oscillations. We solve this problem by associating with each sub-controller a gradient-based parameter training component, which is capable of automatically adapting parameters to achieve their *optimal* values under the current ingress traffic. We evaluate the resulting controller in `ns2` simulations and demonstrate that ABS is able to deal with generic Internet traffic consisting of HTTP sessions, different TCP variants, and non-TCP flows, is robust to changing system dynamics, and is scalable to link capacities and flow populations, all of which make the concept of ABS an appealing and practical buffer sizing framework for future Internet routers.

CHAPTER II

RELATED WORK

1 Next-Generation Congestion Control

1.1 Delay-Dependent Schemes

A large amount of theoretical and experimental work has been conducted to design stable congestion controls for high-speed networks. One example is FAST TCP [49], which utilizes queuing delay, in addition to packet loss, as the primary congestion measurement. It is proved in [50] that FAST TCP is globally asymptotically stable under a single bottleneck link with instantaneous feedback and locally asymptotically stable in a general network with homogeneous feedback delay. However, proper operation of FAST TCP requires reliable computation of the RTT and adaptive tuning of control parameters, both of which are undesirable in realistic networks. Another recently proposed transport protocol is XCP [56], which aims to achieve stable flow adaptation, fare bandwidth sharing, and low packet loss. Instead of simply using packet loss as the congestion signal, XCP generalizes ECN (Explicit Congestion Notification) to indicate the extent of congestion. XCP manages to control efficiency and fairness by using separate controllers inside the routers. Other promising proposed protocols include HSTCP [30], Scalable TCP [61], BIC-TCP [110], LTCP [11], and RCP [26], all of which aim to achieve quick convergence to efficiency, stable rate trajectories, fair bandwidth sharing, and low packet loss.

An entirely different direction in congestion control is to model the network from an optimization or game-theoretic point of view [52, 63, 64, 66, 75]. The original

work by Kelly *et al.* [59, 60] offers an economic interpretation of the resource-user model, in which the entire system achieves its optimal performance by maximizing the individual utility of each end-user. To implement this model in a decentralized network, Kelly *et al.* describe two algorithms (*primal* and *dual*) and prove their global stability in the absence of feedback delay. However, if feedback delay is present in the control loop, stability analysis of Kelly controls is non-trivial and currently forms an active research area [22, 51, 65, 77, 97, 99].

Recall that in Kelly's framework [60, 77], each user $i \in [1, N]$ is given a unique route r_i that consists of one or more network resources (routers). Feedback delays in the network are heterogeneous and directional. The forward and backward delays between user i and resource j are denoted by $D_{ij}^{\rightarrow}$ and $D_{ij}^{\leftarrow}$, respectively. Thus, the round-trip delay of user i is the summation of its forward and backward delays with respect to any router $j \in r_i$: $D_i = D_{ij}^{\rightarrow} + D_{ij}^{\leftarrow}$. Under this framework, Johari *et al.* discretize Kelly's primal algorithm as follows [51]:

$$x_i(n) = x_i(n-1) + \kappa_i\Big(\omega_i - x_i(n - D_i)\sum_{j \in r_i} \mu_j(n - D_{ij}^{\leftarrow})\Big), \tag{1}$$

where κ_i is a strictly positive gain parameter, ω_i can be interpreted as the willingness of user i to pay the price for using the network, and $\mu_j(n)$ is the congestion indication function of resource j:

$$\mu_j(n) = p_j\Big(\sum_{u \in s_j} x_u(n - D_{uj}^{\rightarrow})\Big), \tag{2}$$

where s_j denotes the set of users sharing resource j and $p_j(\cdot)$ is the price charged by resource j. Note that we use a notation in which $D_i = 1$ means immediate (i.e., most recent) feedback and $D_i \geq 2$ implies delayed feedback.

Next, recall that for a homogeneous delay D, system (1)-(2) is locally stable if

[51]:

$$\kappa_i \sum_{j \in r_i} \Big((p_j + p_j' \sum_{u \in s_j} x_u) \Big|_{x_u^*} \Big) < 2 \sin \left(\frac{\pi}{2(2D-1)} \right), \tag{3}$$

where x_u^* is the stationary point of user u and $p_j(\cdot)$ is assumed to be differentiable at x_u^*.

For heterogeneous delays, a combination of conjectures made by Johari *et al.* [51], derivations in Massoulié [77], and the proofs of Vinnicombe [97] suggest that delay D in (3) can be simply replaced with individual delays D_i to form a system of N stability equations; however, the proof exists only for the *continuous* version of (1) and leads to the following necessary stability equation [97]:

$$\kappa_i \sum_{j \in r_i} \Big((p_j + p_j' \sum_{u \in s_j} x_u) \Big|_{x_u^*} \Big) < \frac{\pi}{2D_i}. \tag{4}$$

Inspired by Kelly's optimization framework, one additional method called MaxNet is proposed in [107] and is aimed at improving convergence properties [106] of traditional models of additive feedback. In MaxNet, each user i obtains feedback $\eta_i(t) = \max_{j \in r_i} p_j(t)$ from the most congested router in its path and applies $\eta_i(t)$ to an unspecified control law $x_i(t) = f_i(\eta_i(t))$. Based on the technique developed in [82], the authors prove that MaxNet is locally stable in generic networks with fixed bottleneck assignments if

$$0 < f_i'(\eta_i^*) < \frac{x_i^*}{D_i}, \tag{5}$$

where x_i^* and η_i^* are, respectively, the equilibrium rate and stationary feedback of flow i.

1.2 Delay-Independent Schemes

To the best of our knowledge, the first delay-independent stability condition is due to Vinnicombe, who proposes and examines the following continuous fluid model of

a network with sources operating TCP-like algorithms [98]:

$$\dot{x}_i(t) = \frac{x_i(t-D_i)}{D_i}\big(\alpha_i(t) - \big(\alpha_i(t)+\beta_i(t)\big)\eta_i(t)\big), \tag{6}$$

where $\alpha_i(t) = a\big(x_i(t)D_i\big)^n$, $\beta_i(t) = b\big(x_i(t)D_i\big)^m$, a, b, m, n are constants, network feedback $\eta_i(t) = \sum_{j\in r_i} p_j(n - D^{\leftarrow}_{ij})$, and link price $p_j(t) = (y_j(t)/C_j)^B$ is an approximation of packet loss at link j of capacity C_j and buffer size B. It is proven in [98] that the above controller is locally stable if

$$a(x_i^* D_i)^n < \frac{1}{B}. \tag{7}$$

Setting $n = 0$ and $a < 1/B$, the resulting system achieves delay-independent stability. An additional result is available from [112], where Ying *et al.* considers the following variant of controller (6):

$$\dot{x}_i(t) = \kappa_i x_i(t-D_i)\Big(\frac{1}{x_i^n(t)} - x_i^m(t)\eta_i(t)\Big), \tag{8}$$

where κ_i is a constant. The authors prove that (8) is globally stable regardless of delay in general network topologies if $m + n > B$. This work is similar in spirit to ours; however, the analysis and proposed methods are different.

2 Router Buffer Sizing

The optimal buffer size depends on the target performance constraints (such as link utilization, packet loss rate, and queuing delay). For instance, considering only utilization, buffers should be sufficiently large to prevent the router from idling so that full utilization is achieved. In particular, it is commonly suggested that the buffer size b of a bottleneck router should be at least the product of the output link's capacity C and the average round-trip time R of all incoming TCP sessions, i.e., $b \geq CR$. This

rule-of-thumb is commonly attributed to Villamizar and Song [96] and is deployed in most current large commercial routers [3]. However, the huge amount of memory space required by this rule becomes progressively unrealistic as link speed of the Internet evolves to the magnitude of multiple giga-bps and even tera-bps.

As pointed out by Appenseller *et al.* [3], this classic principle is applicable in scenarios where only synchronized long-lived TCP flows are present. However, Internet core routers are usually utilized by hundreds of thousands of flows, in which case synchronization rarely happens and the aggregate window size process converges to a Gaussian process [3]. Based on this result, they prove that when router buffers are sized according to $b = CR/\sqrt{N}$, link utilization is lower bounded by 98.99%. A completely different result is given by Avrachenkov *et al.* [5], who model buffer sizing as an optimization problem and derive that the optimal buffer size of N unsynchronized flows is $b = (CR)^2/32N^3$. Both results deviate from the rule-of-thumb in that their suggested buffer sizes scale inversely to the number of flows, indicating that all current backbone routers are over-buffered and their memory space and costs can be substantially reduced.

The small-buffer criteria are extended by Enachescu *et al.* [27], who suggest that buffers be as small as $10 - 20$ packets in core routers provided the packet arrival process follows a Poisson distribution. This assumption is enforced by introducing *Paced TCP* [27], where senders evenly spread out-going packets over an RTT. This result is further extended to the model of combined input-output queue [8] and later supported in [87, 103].

Although the assumption of totally asynchronous flows and Poisson arrivals are sound for backbone routers, as pointed out in [25], generic Internet routers are usually accessed by partially synchronized flows. In this case, the minimum buffer require-

ment is shown to be [25]:

$$b = \frac{p(N)CR_e - 2SN(1-p(N))}{2-p(N)}, \tag{9}$$

where R_e is the harmonic mean of the RTTs, S is the MTU, $p(N) = 1-(1-1/N)^{L_N}$ is the fraction of flows that see at least one packet loss, and L_N is the average number of dropped packets during a congestion event.

Besides saturating a given link, Dhamdhere *et al.* [25] propose that minimizing packet loss rate should also be taken into account when sizing router buffers. To accomplish this goal, they develop a buffer management rule based on *Flow Proportional Queuing* (FPQ), in which the loss rate is kept within a threshold value p by increasing the RTTs (or the buffer size) of the flows proportionally to N. Letting R_p and R_q^* respectively be the propagation and required queuing delays of the link, the proposed buffer sizing equation is:

$$b = CR_q^* = K_p^* N - CR_p, \tag{10}$$

where $K_p^* = 0.87/\sqrt{p^*}$ and p^* is the target loss rate. If we consider both utilization and loss constraints, buffer size should be the larger of (9) and (10) and the resulting mechanism is called *Buffer Sizing for Congested Link* (BSCL) [25].

Note that buffer sizing rule (10) suggests that the bottleneck buffer should linearly increase with N, which is in sharp contrast to the aforementioned small-buffer rules [3, 5, 27]. Furthermore, utilizing a generalized AIMD model, Eun and Wang [28] arrive at an interesting conclusion that it is possible to achieve both full utilization and zero loss rate for any *intermediate* buffer size b between $RC/\sqrt{N}$ and RCN. Recent studies by Srikant *et al.* [68] and Dovrolis *et al.* [85] suggest that the optimal buffer size should also depend on the ratio between the input and output links' speed. Specifically, if the ratio is small, buffers of constant sizes $O(1)$ are sufficient for high

utilization and loss loss rate; otherwise, large buffers are required.

Compared to the above schemes that seek to derive an explicit model of buffer size and Internet traffic, another class of methods tries to solve this problem by utilizing a certain implicit relationship between them. Specifically, Shorten *et al.* [92] propose a buffer sizing approach by introducing several sender-side modifications to TCP. They demonstrate that the resulting method, called *Adaptive AIMD*, adapts to any buffer size in the path while preserving fairness and TCP-friendliness between different flows. In addition, Kellett *et al.* [57] formulate the relationship between buffer size and utilization as a sector-bounded nonlinearity and develop an adaptive buffer sizing scheme called *Adaptive Drop Tail* (ADT), whose control equation is given by:

$$b(n) = b(n-1) + K\big(u^* - u(n)\big), \tag{11}$$

where unspecified parameter K needs to satisfy $K = (0, 2/k_2)$ to achieve stability and k_2 is the sector nonlinearity upper bound. However, it is unclear how k_2 is obtained for a given traffic condition and how it can be calculated in real time as the system's dynamics change. Clearly, ADT has to resolve these issues before being used in real Internet routers.

CHAPTER III

EFFECT OF DELAY ON THE PERFORMANCE OF EXISTING CONGESTION CONTROL PROTOCOLS

1 Introduction

In this chapter, we aim to understand how feedback delays of regular end-users, induced by *queuing* at the bottleneck router, affect the behavior of existing congestion control protocols. We do not address the issue of delays arising due to large propagation delays, or due to queuing at non-bottleneck routers. To better understand why delay is a significant factor in the performance of a congestion controller, consider the following illustration. We placed an MPEG-4 video server at Michigan State University and used home DSL clients to stream scalable MPEG-4 FGS video from the server. In the case reported in this work, the client's access link supported up to 512 kb/s on the physical layer, which corresponded to approximately 450 kb/s of IP-layer throughput. As shown in Figure 2 for one representative experiment, both the RTT of the flow and the sending rate of the server exhibited significant fluctuation throughout this ten-minutes streaming session. The increase in the RTT was in response to buffering delays at the DSL link and was directly correlated with occurrences of packet loss. The server used rate-based AIMD congestion control, which frequently increased the rate to over 600 kb/s and then dropped it below 35 kb/s in response to significant packet loss. Clearly, this performance is undesirable for many reasons, including fluctuating video quality, high packet loss, low link utilization, and prohibitively large retransmission delays.

In what follows in this chapter, we first study TCP and demonstrate that its

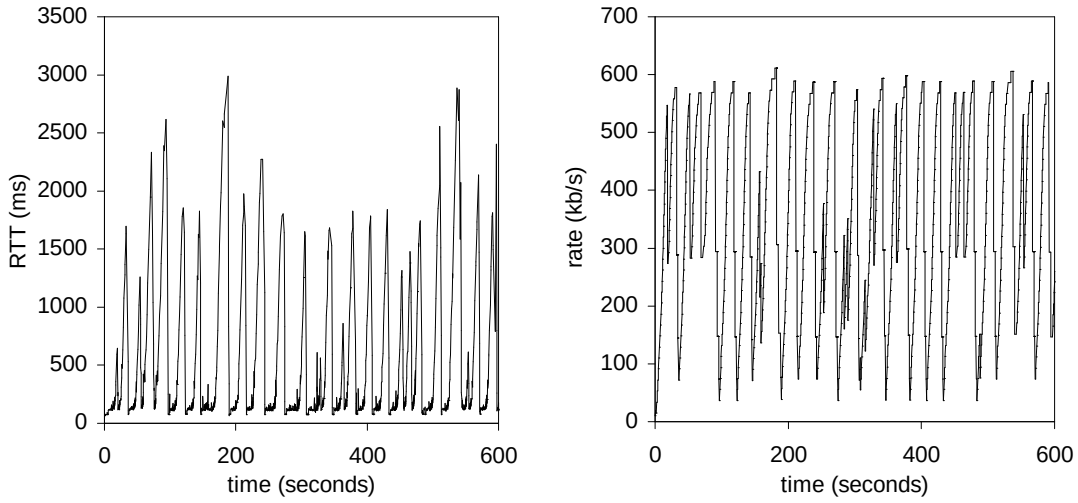


Fig. 2. Video streaming over 512-kb/s residential DSL. Evolution of the RTT (left) and that of the IP-layer sending rate (right).

performance deteriorates as buffering delay becomes large and reaches unacceptable levels (i.e., high rates of oscillations and packet loss). However, we subsequently show that TCP has the *best* delayed performance compared to other protocols including rate-based AIMD, Scalable TCP, and TFRC. Combined, these two results paint a rather bleak picture for current best-effort streaming methods; however, they provide clear insight into a long-standing question of whether window-based protocols do in fact perform best in the current Internet. We conclude the chapter with observations that delayed instability is inherent to all AIMD-friendly classes of control methods and that better algorithms based on AQM feedback may be the only alternative for improving the performance of congestion control in the future Internet.

2 Delayed Behavior of AIMD

Existing congestion controls are broadly classified into two categories: rate-based and window-based, where the former is usually more desirable in video streaming. It is generally observed that controlling rate-based flows is challenging [72, 109] as their

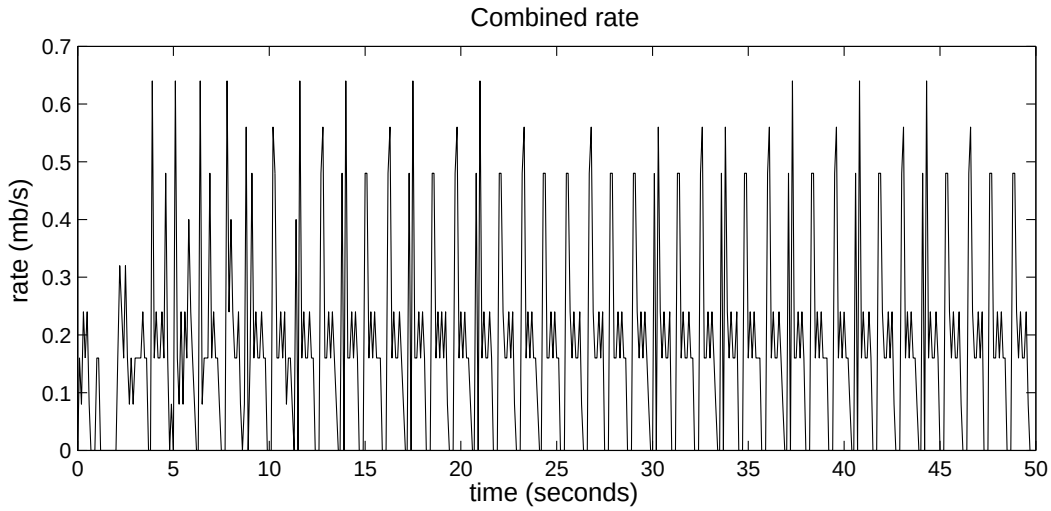


Fig. 3. Combined sending rate of two competing TCP flows.

oscillations and packet loss exhibit worse performance than those of similar window-based mechanisms; however, we are not aware of any quantitative studies that support this statement in the context of streaming or outside of ATM data-link literature. While all TCP-friendly controls oscillate, these oscillations become more pronounced as the delay in the feedback increases (we call this situation "increased instability"), which happens when the flows adjust their current rates based on out-dated feedback and then overreact to packet loss caused by the overshoot.

2.1 Delay-Related Oscillations in TCP

We first consider delayed behavior of TCP. Recall that TCP follows the AIMD policy [19, 47] in its window adjustment, where the sender additively increases its congestion window per positive ACK and multiplicatively decreases it upon each packet loss. When buffers sizes are large (such as in the DSL example shown in the introduction), propagation delay is less of an issue and contributes negligibly to the delayed feedback and/or the dynamics of the whole system. Thus, in the rest of the chapter, we assume

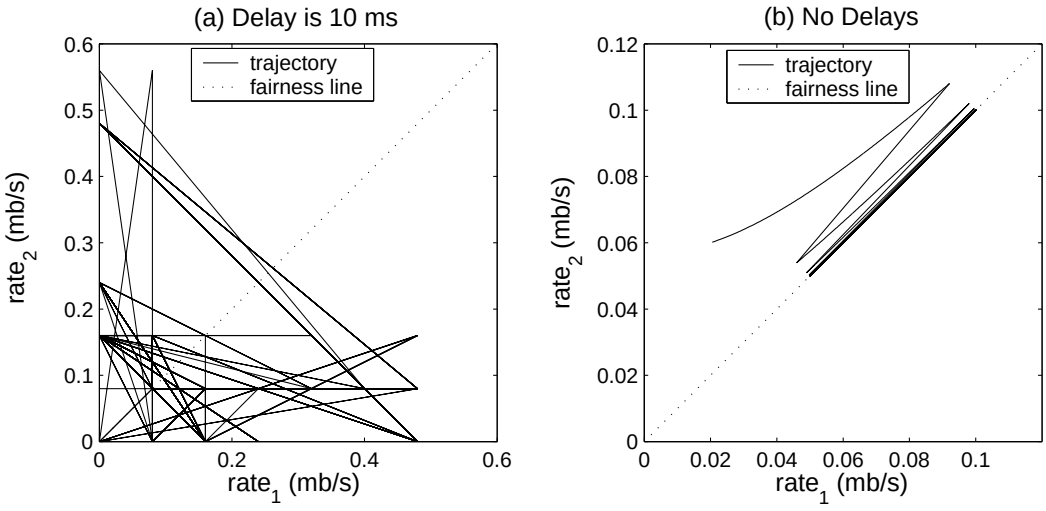


Fig. 4. Trajectories of two TCP flows: (a) 10 ms delay; (b) Instantaneous feedback.

a network configuration with delays arising only due to buffering at the most congested (bottleneck) router of a given path.

In such a model, when the combined sending rate of all flows exceeds the bottleneck capacity, TCP continues increasing its window and queuing the extra data at the bottleneck until congestion is detected. The delay needed for TCP to realize that the buffer is full is directly related to the amount of *bursty* packet loss the flow will suffer and the amount of window-size reduction following the loss.

It is well known that under ideal assumption of instantaneous feedback, all AIMD congestion controls converge to the fair operating point with minimal packet loss [19]. However, when large buffers in the network create delays, TCP becomes more "unstable" and exhibits complex behavior on small time-scales. Consider one such example. We use the `ns2` simulator and examine two competing TCP flows sharing a bottleneck link with 0.2 mb/s bandwidth and 10 ms propagation delay. The buffer size is 20 packets and the sampling rate is 100 ms. Figure 3 depicts the combined rate of these two flows. As seen in the figure, the *average* combined rate is just

below the link's capacity; however, the instantaneous combined rate reaches as high as 0.6 mb/s, which overflows the link and leads to large bursts of packet loss. In response to this loss, the combined rate periodically drops almost to zero, which leads to under-utilization of network resources and amplified oscillations compared to the non-delayed case.

We next examine how fairness between the two flows in Figure 3 is affected by delays. As shown in Figure 4 (a), when the link becomes congested, the flows persistently oscillate around the fairness line and do not maintain fairness on small time-scales. Moreover, the trajectory exhibits no noticeable regularity and appears unpredictable. In contrast, Figure 4 (b) depicts the behavior of two TCP flows under immediate feedback, where the oscillations along the fairness line are relatively small and fairness is maintained at all times.

To understand the delayed behavior of TCP in analytical terms, we next investigate its control equation and derive the amount of lost data as a function of queuing delay D.

2.2 Delayed TCP

It is well known that AIMD mechanisms buffer excess data when the bottleneck link is saturated since feedback delays prevent the sources from adjusting their rates in a timely manner. For sufficiently large feedback delays (i.e., large buffers), end-users eventually overshoot the bottleneck link and experience bursty packet losses due to the excess data sent into the network before congestion is detected. We call this behavior "excessive buffering" and examine its extent in AIMD and other protocols in the rest of this chapter.

We start our investigation with window-based AIMD (e.g., TCP) schemes. Our next result shows that as soon as the bottleneck link is saturated, TCP automatically

switches the growth rate of its window $W(t)$ from linear to $\sqrt{t}$. This naturally leads to a more "conservative" behavior of TCP under non-negligible buffering delays and explains its advantage over the other methods studied in this chapter.

Lemma 1. *After the bottleneck link is saturated, the size of TCP's congestion window grows proportionally to* $\sqrt{t}$.

Proof. Assume discrete time and recall that TCP's congestion window $W(t)$ is increased by $MTU^2/W(t-1)$ upon each positive (non-duplicate) ACK [1]. For clarity of presentation, we express window size in units of packets instead of bytes. Using this notation, $W(t)$ is given by:

$$W(t) = \begin{cases} W(t-1) + 1/W(t-1) & \text{per ACK} \\ \beta W(t-1) & \text{per loss} \end{cases} , \tag{12}$$

where β is the factor of multiplicative decrease and $W(t)$ is congestion window in units of packets. After the link is saturated, TCP increments its window in response to a stream of ACKs coming from the receiver at fixed average rate C pkts/sec, where C is the bottleneck link capacity. This happens because the bottleneck continues buffering some incoming data, while transmitting the remaining packets to the receiver exactly at the rate of C pkts/sec. Once these packets arrive at the receiver, they generate a stream of ACKs at the same packet-rate C. It is common to approximate (12) with a fluid model, in which the ACKs arrive as a fluid at the exact rate C. Then, we can re-write (12) using a differential equation:

$$\frac{dW}{dt} = \frac{C}{W} . \tag{13}$$

Re-organizing (13) and integrating both sides, we get:

$$W(t) = \sqrt{2Ct + \delta} , \tag{14}$$

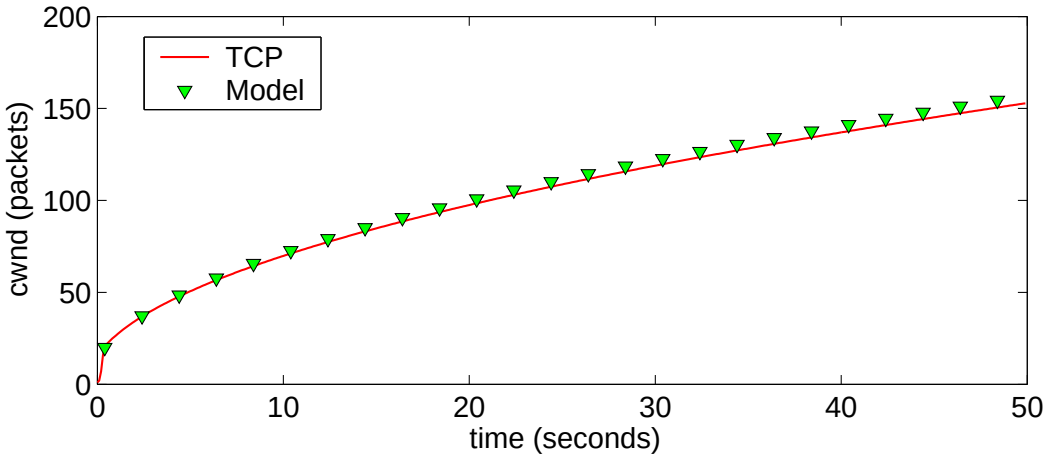


Fig. 5. The congestion window of a single TCP flows under delay.

where $\delta = W(0)^2$ is the integration constant. □

We next consider an `ns2` simulation to validate the conclusion of Lemma 1. In this simulation, we run a single TCP flow over a bottleneck link with an infinitely large buffer. The packet size is $1,040$ bytes, the bottleneck link capacity $C = 2$ mb/s (244 pkts/sec), and the round-trip propagation delay is 60 ms. Using $C = 244$ pkts/sec in (14), observe from Figure 5 that `ns2` simulations match the model very well.

As demonstrated above, TCP overshoots the bottleneck link after saturation and its congestion window tends to infinity provided that the bottleneck queuing delay allows so. Note, however, that the sending rate $r(t)$ of TCP does not grow to infinity (and in fact converges to link capacity C as we show below) since the RTT also increases when packets start being queued at the bottleneck link. Nevertheless, the amount of extra data sent into the link (all of which gets lost *prior* to TCP's reaction to the actual losses) is non-negligible as we show in the next result.

Lemma 2. *The aggregated amount of lost data in window-based AIMD during each*

overshoot is proportional to the square root of the buffering delay $\sqrt{D}$.

Proof. Assume the time starts at $t = 0$ when the bottleneck link is about to overflow. For each received ACK after $t = 0$, a TCP source sends out $1 + 1/W(t)$ packets, which means that the amount of "extra" data injected into the network is $1/W(t)$ packets per ACK. As discussed in the proof of Lemma 1, the ACKs are fed back to the sender at rate C pkts/sec. Then, the amount of extra data $S(t)$ sent into the link during the segment $[0, t]$ can be modeled by a simple recurrence:

$$S(t) = S(t-1) + \frac{1}{W(t-1)}, \tag{15}$$

where discrete time t is given in ACKs and starts from the point of bottleneck saturation. Converting the above into a differential equation and shifting time to seconds:

$$\frac{dS(t)}{dt} = \frac{C}{W(t)}\ , \tag{16}$$

where congestion window $W(t)$ is given in (14). Expanding $W(t)$:

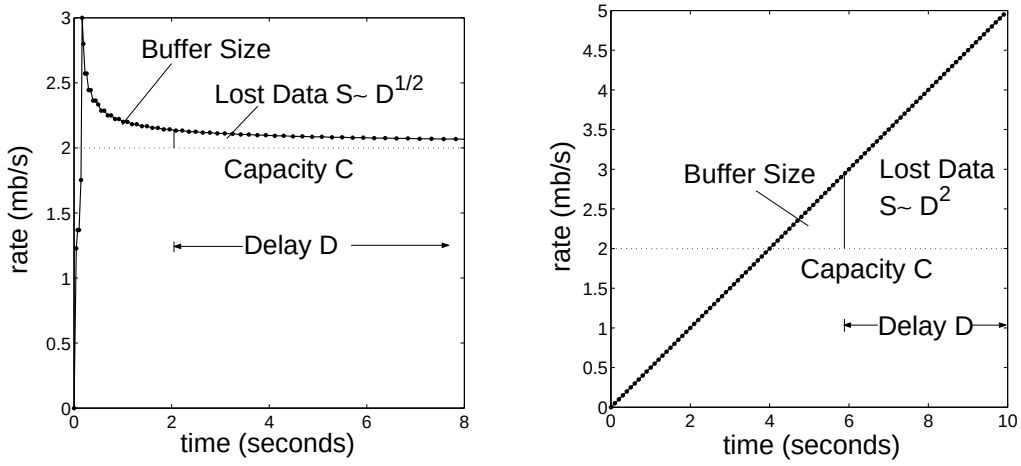
$$\frac{dS(t)}{dt} = \frac{C}{\sqrt{2Ct+\delta}}, \tag{17}$$

where $\delta = W(0)^2$ is again the square of the congestion window just before the link overflows at time $t = 0$. Re-organizing the terms in (17) and integrating both sides, we have:

$$S(D) = \int_0^D \frac{C}{\sqrt{2Ct+\delta}} dt. \tag{18}$$

Solving this integral, we get $S(D) \sim \sqrt{2CD+\delta}$. □

We now return to the sending rate $r(t)$ of TCP. Recall that after the bottleneck link is saturated, TCP sends out $1 + 1/W(t)$ packets per ACK and ACKs arrive at rate C. According to (14), congestion window W tends to infinity as t becomes large.



(a) (b)

Fig. 6. (a) Rate adjustment of window-based protocols under delays. (b) Rate adjustment of rate-based protocols under delays.

Thus, $1 + 1/W(t) \to 1$ and TCP eventually sends out exactly one packet per ACK and converges its rate $r(t)$ to C as time progresses.

Consider an `ns2` illustration in Figure 6 (a), where we use the same configuration as in Figure 5 except that we now disable slow start to better simulate TCP's behavior in congestion avoidance. Notice in the figure that, after the bottleneck link becomes full, the increase in TCP's rate is slowed down and its $r(t)$ eventually converges to capacity C.

2.3 Delayed Rate-Based AIMD

We next examine a class of *rate-base* AIMD flows, in which the control actions take places once per RTT instead of once per ACK. Note that both window-based and rate-based AIMD perform essentially the same until the point at which their sending rates start to exceed capacity C; after that, their performance becomes drastically different. The explanation of this phenomenon is simple as rate-based methods must

battle the various difficulties in *accurately* and *timely* estimating the RTT and noticing its increase in response to a growing buffer at the bottleneck (this information is automatically supplied to window-based methods through positive ACKs). Unfortunately, a closed-form solution to the exact queuing model coupled with end-flow control equations does not exist for both rate-based AIMD and TFRC, even when we assume that the delay in obtaining RTT samples is negligible. The case becomes more complicated when the RTT feedback is delayed and/or smoothed with an exponential filter.

In this section, we only solve the simpler case of rate-based AIMD and TFRC in which the RTT is not accurately tracked by the source (i.e., is perceived to remain more or less constant until packet loss is detected) and leave the more extensive analysis of variable RTTs in future work.

Lemma 3. *Under constant RTT, the amount of lost data in rate-based AIMD during each overshoot is proportional to* D^2.

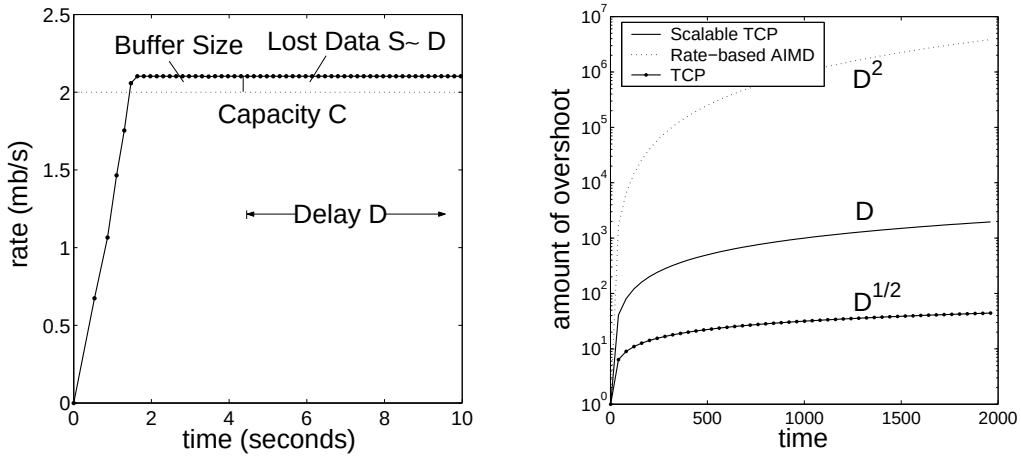
Proof. The equation for rate-based AIMD is given by:

$$r(t) = \begin{cases} r(t - RTT) + \alpha & \text{per RTT} \\ \beta r(t - RTT) & \text{per loss} \end{cases}, \tag{19}$$

where $r(t)$ is the sending rate at time t. The source increases its sending rate by constant α per RTT and reduces it by a factor of β upon each packet loss. Taking feedback delays into account, the increase phase of (19) can be written in the fluid sense as following:

$$\frac{dr(t)}{dt} = \frac{\alpha}{RTT(t - \Delta)}, \tag{20}$$

where Δ is the delay needed to drain parts of the buffer for the source to recognize the increase in the RTT and $RTT(t) = q(t)/C$ is the queue size at time t divided



(a) (b)

Fig. 7. (a) Rate adjustment of Scalable TCP under delay. (b) Comparison of lost data per overshoot in TCP, Scalable TCP, and rate-based AIMD.

by capacity C. Closed-form solution to this model does not exist even when the feedback delay is zero (i.e., $\Delta = 0$). We abandon this direction and study a model with constant RTT, which often happens in practice when $\Delta \geq D$ and the flow cannot realize that its RTT has increased *until after* it has drained the bottleneck queue (by which time packet loss has occurred and any attempts to adjust the rate are too late from the control-theoretic view).

Solving (20) with constant RTT, we obtain that the sender's rate is a linear function of time (i.e., $r(t) \sim t$) and the amount of lost data by time t is:

$$S(t) = \int_0^t (r(u) - C)du = \int_0^t r(u)du - Ct, \tag{21}$$

where time $t = 0$ is again the instant when the buffer is about to overflow. From (21), we obtain that $S(t) \sim t^2$ and $S(D) \sim D^2$. □

This situation is illustrated in Figure 6 (b), in which the amount of overshoot S grows quadratically as a function of time needed for the source to react to packet loss.

Taking into account *variable* RTT, numerical solutions to the exact queuing model (20) show that for a certain amount of time immediately following the overshoot of C, $r(t)$ behaves as a linear function; however, the remaining increase in $r(t)$ is only logarithmic. Nevertheless, both cases show that rate-based AIMD grows its sending rate to infinity under a sufficiently large delay.

Compared to window-based AIMD methods, this growth in $r(t)$ is clearly a problem and supports the observation that rate-based AIMD flows experience more packet loss and higher oscillations than their window-based counterparts [72, 109].

3 Next-Generation TCP

Recent research efforts to design better congestion controls for high-bandwidth networks have led to the development of next-generation TCPs – High-speed TCP (HSTCP) [30] and Scalable TCP [61] – which incorporate simple and easily deployable changes to classical TCP. Since HSTCP is similar to Scalable TCP, we only consider the latter and examine its behavior under delayed feedback.

Lemma 4. *The amount of lost data in Scalable TCP during each overshoot is proportional to D.*

Proof. Recall that Scalable TCP relies on the following binary-feedback controller [61]:

$$W(t) = \begin{cases} W(t-1) + 0.01 & \text{per ACK} \\ .875W(t-1) & \text{per loss} \end{cases}, \tag{22}$$

where $W(t)$ is the size of congestion window at time t. Notice that after each RTT, congestion window $W(t)$ is increased by a factor of 1.01 since for each RTT, the number of arriving ACKs is at most equal to the window size (i.e., Scalable TCP is an MIMD controller).

Similar to TCP, the sending rate of Scalable TCP does not grow at the same pace as the window size because of the increased RTT. After the bottleneck link is fully utilized, the source sends out 1.01 packets per ACK (i.e., its rate $r(t) = 1.01C$ exceeds bottleneck capacity by a fixed fraction). Thus, the amount of excess data sent per ACK is fixed, i.e., 0.1 packets, and the total overshoot S grows linearly with time, i.e., $S \sim D$. □

Figure 7 (a) demonstrates the buffering behavior of Scalable TCP in `ns2`. For convenience of visualization, we change STCP's window increase step size from 0.01 to 0.05 such that the difference between the steady-state flow rate and capacity C is easy to identify. As the figure shows, simulations match the discussion in Lemma 4 very well.

To summarize the results, Figure 7 (b) shows a comparison of the amount of lost data among all three methods. Notice that window-based self-clocking in TCP and Scalable TCP are powerful mechanisms that prevent the sending rates of these flows from growing to infinity when the bottleneck link becomes saturated.

4 TFRC

TFRC (TCP-Friendly Rate Control) (e.g., [31, 81]) has become a de-facto standard for multimedia applications. Instead of immediately responding to congestion in a manner like TCP, TFRC gradually adjusts its rate if the congestion persists. Recall that TFRC uses a discrete TCP-friendly equation and directly adjusts its rate based on the latest measurement of packet loss and RTT:

$$r(n) = \frac{MTU}{\sqrt{p(n-\Delta_1)}RTT(n-\Delta_2)} \ , \tag{23}$$

where MTU is the maximum transmission unit, $p(n)$ is the long-term average packet loss computed at time n, and $RTT(n) = q(n)/C$ is the round trip delay seen inside the router at time n. Since both packet loss and RTT are fed back from the network, their values are retarded by the corresponding feedback delays Δ_1 and Δ_2.

Again assuming that the source maintains a constant RTT until the first loss is detected (i.e., either because $\Delta_2 > D$ or the smoothing filter on RTT exhibits slow convergence), TFRC's behavior is stated in the following lemma.

Lemma 5. *Under constant RTT, the amount of lost data in TFRC during each overshoot is proportional to* D^2.

Proof. Assume at time $t = 0$ the buffer is about to overflow and the average long-term packet loss up to that point is strictly positive. Let $M(n) > 0$ be the number of lost packets up to time n and $T(n)$ be the total number of transmitted packets up to time n. Modeling $p(n)$ as the long-term average loss[1], we get:

$$p(n) = \frac{M(n)}{T(n)},\ t \geq 0, \tag{24}$$

where $M(n) = M$ is constant since the source has not detected any *new* buffer overflows and continues to perceive the network as uncongested. Since the change in $T(n)$ is simply rate $r(n)$, we can write the following continuous-time model:

$$\frac{dT(t)}{dt} = r(t), \tag{25}$$

where dT/dt represents the number of packets sent per time unit. Combining this

[1]To keep the problem tractable, we simplify several aspects in TFRC's computation of packet loss.

with (23)-(24) and assuming instantaneous feedback, we have:

$$\frac{dT(t)}{dt} = \frac{\omega\sqrt{T(t)}}{\sqrt{M}}, \tag{26}$$

where $\omega = MTU/RTT$ is a constant. Reorganizing (26) and integrating both sides, we get $T(t) \sim t^2$ and $r(t) \sim t$. Thus, the amount of lost data also increases quadratically with delay D. □

We next explore control-theoretic stability of TFRC under the assumption that an AQM-enabled router can feed back the exact value of the most-recent packet loss $p(n)$. Such AQM support no longer requires TFRC to smooth the long-term packet loss obtained from the receiver and may potentially improve TFRC's performance. However, as our next lemma shows, this is not the case.

Lemma 6. *Under AQM feedback and constant RTT, TFRC can only be stabilized at points x^* that incur no less than 33% packet loss. For other values of packet loss, TFRC cannot be stabilized and oscillates.*

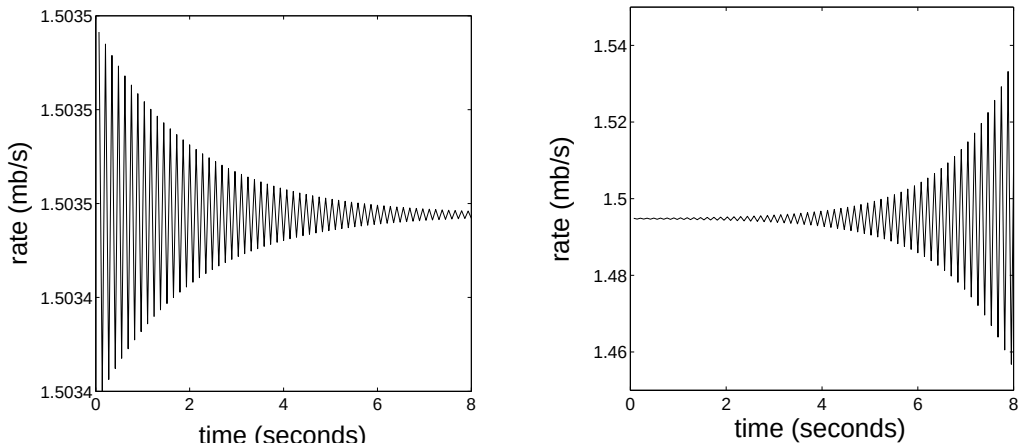
Proof. Write a TFRC control equation for a single flow and immediate AQM feedback $p(n) = (r(n) - C)^+/r(n)$:

$$r(n) = \frac{\omega\sqrt{r(n-1)}}{\sqrt{r(n-1) - C}}\ , \tag{27}$$

where $\omega = MTU/RTT$ is a constant and we assume that $r(n-1) > C$. The only non-negative stationary point of this system is given by:

$$r^* = \frac{C + \sqrt{C^2 + 4\omega^2}}{2}\ . \tag{28}$$

We next check local stability of TFRC at r^* and derive its average loss in the sta-



(a) (b)

Fig. 8. (a) Behavior of TFRC for different stationary points: (a) $r^* = 1.503C$; (b) $r^* = 1.494C$.

tionary state. Linearizing (27) in r^*, we get:

$$\left.\frac{\partial r(n)}{\partial r(n-1)}\right|_{r^*} = \left.\frac{-\omega C}{2\sqrt{r}(r-C)^{3/2}}\right|_{r^*} = \frac{-C}{2(r^*-C)}. \tag{29}$$

Recall that system (27) is stable at r^* if the absolute value of (29) is less than one, which translates into $|\frac{C}{2(r^*-C)}| < 1$ where r^* is in (28). Solving this inequality, we have:

$$r^* > \frac{3C}{2}. \tag{30}$$

This means that system (27) may be stable in the stationary point only at the expense of at least 33% of the packets being dropped. □

According to (28), the stationary point x^* can be tuned by properly choosing constant ω. Consider a simulation of (27) in Figure 8 with two different stationary points. In Figure 8(a), we set capacity C to 1 mb/s, the MTU to 1,500 bytes, and the RTT to 13.8 ms (ω = 870 kb/s). Under these conditions, the stationary point x^* is 1,503 kb/s and the flow clearly converges to x^* after decaying oscillations. In

Figure 8(b), we adjust the RTT to 14 ms such that $x^* = 1,494$ kb/s. As the figure shows, the system diverges even when started in a close vicinity of the stationary point. Thus, within the operating range of most applications (i.e., packet loss below 33%), AQM-TFRC cannot be stabilized.

5 Discussion

It is possible that eventually the Internet will adopt a class of AQM-based mechanisms that are capable of providing asymptotically stable congestion control to end-flows. In such a case, we find that a wide variety of algorithms, including XCP [56], RCP [26], and Kelly controls [60, 66, 75] will be able to supply rate-based, oscillation-free virtual channels. While the issue of designing oscillation-free, rate-based congestion control for best-effort networks remains open, we find that window-based protocols are expected to become more popular as they offer better (albeit far from ideal) performance under delay and an easy-to-implement platform being part of the TCP/IP protocol stack. On a bigger scale, we believe that more effort should be put into scalable extensions to control methods that are *provably* stable under arbitrary delay in the control-theoretic sense and that their presentation to the streaming community should become more "digestible." We next present our work in this direction and demonstrate that *rate-based* control methods can not only be oscillation-free, but also provably stable and fair under *heterogeneous* end-user feedback delays.

CHAPTER IV

THEORETICAL FOUNDATION

1 Introduction

In this chapter, we gain a deeper insight into stability of single-link congestion control systems under diagonal delays. Most max-min systems (e.g., MKC [115], RCP [26], and XCP [56]) can be linearized to the following shape (more on this below):

$$x_i(n) = \sum_{j=1}^{N} a_{ij} x_j(n - D_i), \tag{31}$$

where a_{ij} are some constants, N is the number of flows, and $x_i(n)$ and D_i are, respectively, the sending rate and round-trip time of user i. Using this model, we first derive a sufficient stability condition of (31) to be $||A||_2 < 1$, where $A = (a_{ij})$ is the coefficient matrix of the system and matrix norm $||.||_2$ is induced by the L^2 vector norm. Subsequently, we prove that this result actually extends to any matrix norm induced by a *monotonic* vector norm (which subsumes all standard vector norms, such as $||.||_1$, $||.||_2$, $||.||_\infty$, and $||.||_\infty^w$). Moreover, we prove that a special norm $||A||_s = \inf_{W \in \mathcal{P}^*} ||WAW^{-1}||_2$ (where $\mathcal{P}^*$ is the set of all positive diagonal matrices) is a monotonic induced norm and further generalize the sufficient stability condition of system (31) to $||A||_s < 1$, whose necessity is also indicated by simulations. Armed with these results, we identify several classes of systems that are stable under diagonal delays if and only if they are stable under undelayed feedback. We also discuss and verify obtained results using Matlab simulations.

In the following section, we present modeling assumptions of single-link conges-

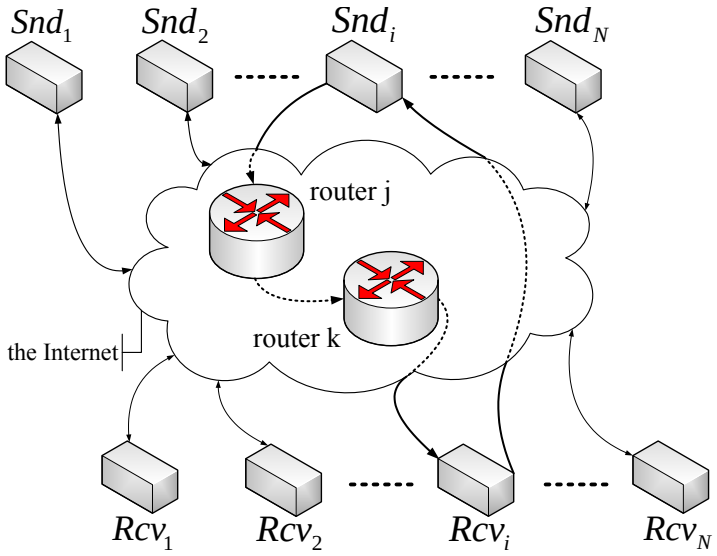


Fig. 9. Model of the network and directional feedback delays.

tion control and existing results in the area of delay-independent stability.

2 Background

2.1 Modeling Single-Link Congestion Control

Assume a network with N users accessing a single bottleneck link. Delays in network feedback arise from both the transmission/propagation time along the data links and the queuing delays at each of the intermediate routers. Consider an illustration in Figure 9, where routers j and k are on the path of sender (user) i. The time lag for a packet to travel from sender i to router j is denoted by forward delay $D_{ij}^{\rightarrow}$, while the delay from router j to the receiver and subsequently from the receiver back to the sender is denoted by backward delay $D_{ij}^{\leftarrow}$. It is clear that the sum of directional delays with respect to each router is the round trip delay of user i, i.e., $D_i = D_{ij}^{\rightarrow} + D_{ij}^{\leftarrow} = D_{ik}^{\rightarrow} + D_{ik}^{\leftarrow}$.

Each acknowledgement packet of flow i carries certain network feedback $p(n)$, which is continually computed by the bottleneck router as a function of the combined

incoming rate of all flows, i.e.,

$$p(n) = g\Big(\sum_{j=1}^{N} x_j(n - D_j^{\rightarrow})\Big). \tag{32}$$

This feedback is utilized by each source i to update its sending rate $x_i(n)$ according to the following control rule:

$$x_i(n) = f_i\big(p(n - D_i^{\leftarrow})\big), \tag{33}$$

where function $f_i(.)$ is assumed to be differentiable in the equilibrium point.

Note that (32)-(33) usually forms a nonlinear system, whose local stability can be studied by linearizing the system in the equilibrium point $\mathbf{x}^*$. Denote by $A = (a_{ij})$ the corresponding Jacobian matrix, then the linearized system is described as follows:

$$x_i(n) = \sum_{j=1}^{N} a_{ij} x(n - D_j^{\rightarrow} - D_i^{\leftarrow}), \tag{34}$$

where $a_{ij} = \partial f_i / \partial x_j|_{\mathbf{x}^*}$.

The following result transforms stability analysis of (34) to that of an equivalent system (31).

Lemma 7. *System (34) is stable under all heterogeneous directional delays $D_i^{\rightarrow}$ and $D_i^{\leftarrow}$ if and only if system* (31) *is stable under all round-trip delays D_i.*

We omit the proof of this lemma for brevity and note that a similar result is derived for continuous-time systems in [77]. Compared to (34), system (31) has a simpler shape and is more amenable to analysis. Thus, in the rest of this chapter, we focus our study on system (31) and keep in mind that our results also apply to (34).

2.2 Existing Stability Results

In this work, we are interested in delay-independent stability as defined below of a given dynamical system.

Definition 1. *We call a system stable independent of delays if neither its control gain nor stability condition explicitly involves delays.*

It is well-known that system (31) under zero delay is stable if and only if the spectral radius $\rho(A) < 1$ [45]. When delay is introduced into the control loop, stability analysis of the resulting system becomes more complicated. The most recent result in this direction is presented in [115], which proves that (31) with a *symmetric* Jacobian A is stable regardless of delay if and only if $\rho(A) < 1$.

One more generic version of this problem is to study stability of the system under arbitrary delay $D_{ij} > 0$, i.e.,

$$x_i(n) = \sum_{j=1}^{N} a_{ij} x_j(n - D_{ij}). \tag{35}$$

We note that in the last equation each feedback is delayed by D_{ij} time units instead of a round-trip delay D_i as in (31). Thus, stability conditions of system (35) are sufficient, but not necessary for system (31).

The convergence property of (35) is initially studied by Chazan and Miranker [18] in the context of asynchronous iteration and the first sufficient and necessary condition is due to Bertsekas and Tsitsiklis [10], who prove that (35) is stable for all uniformly-bounded and time-varying delays $D_{ij}(n)$ if $\rho(|A|) < 1$ and unstable under a certain set of delays $D_{ij}(n)$ if $\rho(|A|) \geq 1$. The same result is later obtained by Kaszkurewicz *et al.* [53] using a different technique.

A slightly stronger version of this result is available in [95], which first introduces the following terminology.

Definition 2 ([95])**.** *Time delays $D_{ij}(n)$ are admissible if:*

$$\lim_{n\to\infty} n - D_{ij}(n) = \infty, \ \forall i, j \tag{36}$$

and regulated (uniformly bounded) if $0 \leq D_{ij}(n) < \bar{D}, \ \forall i, j$ for some non-negative constant $\bar{D}$ that is independent of n.

Then, Su *et al.* [95] prove that system (35) is stable for all admissible delays $D_{ij}(n)$ when $\rho(|A|) < 1$ and unstable for a certain sequence of regulated delays $D_{ij}(n)$ with $\bar{D} = 1$ when $\rho(|A|) \geq 1$.

In addition, Kaszkurewicz *et al.* provide an alternative way of verifying condition $\rho(|A|) < 1$ by showing that it is equivalent to the existence of a positive diagonal matrix W such that $||W^{-1}AW||_\infty < 1$. This is a consequence of [55, Appendix 2]:

$$\rho(|A|) = \inf_{W\in\mathcal{P}^*} ||W^{-1}AW||_\infty, \tag{37}$$

where $\mathcal{P}^*$ is the set of all positive, diagonal matrices.

However, for system (31), condition $\rho(|A|) < 1$ is too strong and is not necessary. One example is given in [115], which demonstrates that MKC in the form of (31) may be stable even though $\rho(|A|) > 1$. The relationship between stability conditions of (31) and (35) under different types of delays is illustrated in Fig. 10. As shown in the figure, stability of the system under zero delay and arbitrary delays D_{ij} has been well studied; however, understanding of (31) under diagonal delays D_i is lacking in the current picture. Thus, in the rest of this chapter, we fill in this void and investigate conditions under which system (31) achieves delay-independent stability.

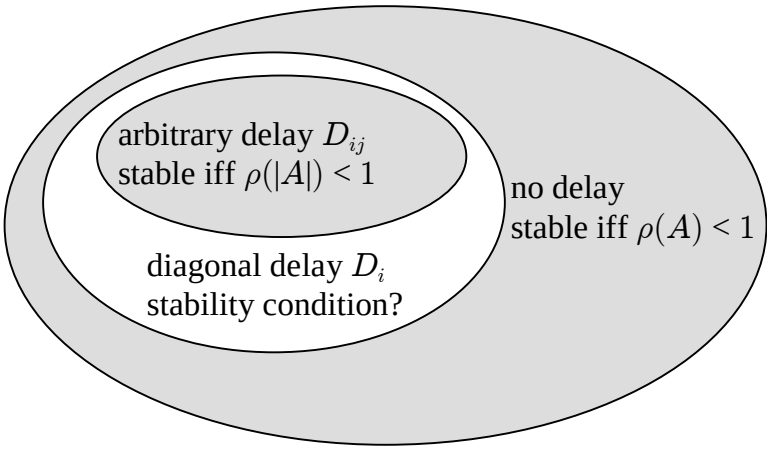


Fig. 10. Illustration of the current research status of delay-independent stability of system (31) under different types of delays.

3 Main Results

3.1 Induced Matrix Norms

We start by recalling definitions of induced matrix norms, which are used later.

Definition 3 ([45]). *Matrix norm* $||.||$ *induced by or subordinate to a given vector norm* $||.||$ *is defined as following:*

$$||A|| = \sup_{x \neq 0} \frac{||Ax||}{||x||}. \tag{38}$$

The following properties of induced matrix norms are available from [45].

Property 1. *Any induced matrix norm* $||.||$ *satisfies inequality* $||Ax|| \leq ||A|| \cdot ||x||$.

Property 2. *For any matrix* A *and an arbitrary induced matrix norm* $||.||$*, we have* $\rho(A) \leq ||A||$.

To better understand Definition 3, consider the following commonly used induced matrix norms: spectral norm $||A||_2 = \sqrt{\rho(A^*A)}$ (where A^* is the conjugate transpose of A) induced by the L^2 vector norm, maximum absolute column sum norm $||A||_1 =$

$\max_j(\sum_i |a_{ij}|)$ induced by the L^1 vector norm, maximum absolute row sum norm $||A||_\infty = \max_i(\sum_j |a_{ij}|)$ induced by the L^∞ vector norm, and weighted maximum norm $||A||_\infty^w = \max_i(\sum_j |a_{ij}|w_j)/w_i$ $(w > 0)$ induced by the weighted infinity vector norm $||x||_\infty^w$. We refer interested readers to [45] for more details.

3.2 First Sufficient Condition

We start with the following result.

Theorem 1. *If A is symmetric and stable, system (31) is stable regardless of delays D_i.*

Proof. Applying the z-transform to system (31), we obtain:

$$\mathbf{H}(z) = ZA\mathbf{H}(z), \tag{39}$$

where $Z = \text{diag}(z^{-D_i})$ is a diagonal matrix and $\mathbf{H}(z)$ is the vector of z-transforms of each flow rate x_i: $\mathbf{H}(z) = \big(H_1(z), H_2(z), \cdots, H_N(z)\big)^T$. Notice that system (31) is stable if and only if all poles of its z-transform $\mathbf{H}(z)$ are within the unit circle in the z-plane. To examine this condition, re-organize the terms in (39):

$$(ZA - I)\,\mathbf{H}(z) = 0. \tag{40}$$

Next notice that the poles of $\mathbf{H}(z)$ are simply the roots of:

$$\det(ZA - I) = 0. \tag{41}$$

Thus, ensuring that all roots of (41) are inside the open unit circle will be both sufficient and necessary for system (31) to be stable. Bringing in notation $\mathcal{F}(z) =$

$\det(ZA - I)$, we can re-write $\mathcal{F}(z)$ as following:

$$\begin{aligned} \mathcal{F}(z) &= \det(Z[A - Z^{-1}I]) \\ &= \det(Z)\det(A - Z^{-1}). \end{aligned} \tag{42}$$

Noticing that $\det(Z)$ is strictly non-zero for non-trivial z, we can reduce (41) to:

$$\mathcal{F}(z) = \det\big(A - Q(z)\big) = 0, \tag{43}$$

where $Q(z) = \mathrm{diag}(z^{D_i})$.

To prove that all roots of (43) lie in the open unit circle, we suppose in contradiction that there exists a root $|z_0| \geq 1$ such that $\mathcal{F}(z_0) = 0$. Denote by B matrix $Q(z_0)$. Following [80] and using basic matrix algebra, it is easy to have that there exists a non-zero vector v such that $Av = Bv$. For symmetric matrices, we can write $||A||_2 = \rho(A) < 1$ and:

$$||A||_2 = \sup_{x \neq 0} \frac{||Ax||_2}{||x||_2} \geq \frac{||Av||_2}{||v||_2} = \frac{||Bv||_2}{||v||_2}, \tag{44}$$

where $||.||_2$ in application to vectors is a standard L^2 norm.

Since B is diagonal with $|b_{ii}| = |z_0|^{D_i} \geq 1$, Bv is simply a vector $(v_1 b_{11}, \ldots, v_N b_{NN})^T$ and we can express vector norm L^2 using the following:

$$\frac{||Bv||_2}{||v||_2} = \frac{\left(\sum_{i=1}^{N} |v_i|^2 |b_{ii}|^2\right)^{1/2}}{\left(\sum_{i=1}^{N} |v_i|^2\right)^{1/2}} \geq \frac{\left(\sum_{i=1}^{N} |v_i|^2\right)^{1/2}}{\left(\sum_{i=1}^{N} |v_i|^2\right)^{1/2}} = 1. \tag{45}$$

Thus, we get that both $||A||_2 \geq 1$ and $||A||_2 < 1$ must be satisfied simultaneously, which is a contradiction. This means that no $|z_0| \geq 1$ can be a root of $\mathcal{F}(z)$ and that any heterogenous systems with a symmetric stable matrix A is stable under arbitrary delay. $\square$

Note that the above reasoning indicates symmetry of A is not necessary and leads us to the following generalization.

Corollary 1. *If Jacobian A satisfies $||A||_2 < 1$, system (31) is stable for all delays D_i.*

3.3 Weaker Sufficient Conditions

We next extend the requirement of L^2 norm in the last result to any *monotonic* vector norm, which is defined below.

Definition 4 ([7])**.** *If a vector norm $||.||$ on R^n satisfies the following inequality:*

$$||x|| \leq ||y|| \text{ for all } x, y \in R^n \text{ such that } |x| \leq |y|, \tag{46}$$

we call this norm monotonic.

This allows us to prove the following theorem.

Theorem 2. *If there exists a monotonic vector norm $||.||_\alpha$ such that induced matrix norm $||A||_\alpha < 1$, system (31) is stable regardless of delays D_i.*

Proof. We utilize the technique in the proof of Theorem 1, whose goal is to show that $||Bv||_\alpha / ||v||_\alpha \geq 1$ for all v. Again, notice that $Bv = (v_1 b_{11}, \ldots, v_N b_{NN})^T$ and that $|b_{ii}| \geq 1$. Due to the monotonicity of the norm and the fact that $|v_i| \leq |b_{ii} v_i|$, we directly get $||v||_\alpha \leq ||Bv||_\alpha$. The remaining proof is similar to that of Theorem 1. □

The following result provides a systematic way for generating monotonic induced matrix norms.

Theorem 3. *Matrix norm $||A||_2^w = ||WAW^{-1}||_2$ for any non-singular diagonal matrix $W = diag(w)$ is a monotonic induced norm.*

Proof. First write:

$$||A||_2^w = \sup_{x \neq 0} \frac{||Ax||_2^w}{||x||_2^w} = \sup_{x \neq 0} \frac{||WAx||_2}{||Wx||_2}. \tag{47}$$

Next, we need to show that $||x||_2^w = ||Wx||_2$ is monotonic with respect to x. In other words, we must show that for any two vectors x_1 and x_2 such that $|x_1| \leq |x_2|$, $||Wx_1||_2 \leq ||Wx_2||_2$. This directly follows from the fact that $|Wx_1| \leq |Wx_2|$ for any non-singular diagonal W and from monotonicity properties of $||.||_2$ in application to vectors. □

Then, a more generic sufficient stability condition is easy to derive from Theorem 2.

Corollary 2. *The following is sufficient for system (31) to be stable for all delays D_i:*

$$||A||_s = \inf_{W \in \mathcal{P}^*} ||WAW^{-1}||_2 < 1, \tag{48}$$

where $\mathcal{P}^*$ is the set of all positive diagonal matrices.

We next show that spectral norm in (48) is weaker than infinity norm, which is used in (37) as the sufficient and necessary condition for stability of system (35).

Theorem 4. *For any matrix A, we have $||A||_s \leq \rho(|A|) = \inf_{W \in \mathcal{P}^*} ||WAW^{-1}||_\infty$.*

Proof. Denote by $D = |A|$ the absolute value of A and observe that:

$$\begin{aligned} ||WAW^{-1}||_2 &= \sqrt{\rho(W^{-1}A^*WWAW^{-1})} \\ &= \sqrt{\rho(A^*W^2AW^{-2})} \\ &\leq \sqrt{\rho(D^*W^2DW^{-2})} \\ &= ||WDW^{-1}||_2, \quad W \in \mathcal{P}^* \end{aligned} \tag{49}$$

where A^* is the conjugate transpose of A. Next, since D is a non-negative matrix,

we immediately obtain from [54, Lemma 2.7.25] that:

$$\inf_{W\in\mathcal{P}^*} ||WDW^{-1}||_2 = \rho(D). \tag{50}$$

Completing the chain of arguments, we get:

$$\inf_{W\in\mathcal{P}^*} ||WAW^{-1}||_2 \leq \inf_{W\in\mathcal{P}^*} ||WDW^{-1}||_2 = \rho(|A|) \tag{51}$$

for all matrices A. □

3.4 Delay-Independent Stable Matrices

In this subsection, we apply results obtained so far and identify several classes of matrices that are stable under diagonal delays if and only if they are stable under zero delay, i.e., $\rho(A) < 1$.

We first examine the class of normal matrices $\mathcal{N}$, which are defined as the set of matrices A for which $AA^* = A^*A$, where A^* is the conjugate transpose of A. Normal matrices include symmetric (i.e., $a_{ij} = a_{ji}$), skew-symmetric (i.e., $a_{ij} = -a_{ji}$), Hermitian (i.e., $A^* = A$), skew-Hermitian (i.e., $A^* = -A$), circulant, and unitary matrices (i.e., $A^* = A^{-1}$).

Lemma 8. *If $A \in \mathcal{N}$, A is stable under diagonal delays D_i if and only if $\rho(A) < 1$.*

Proof. First notice that if matrix A is normal, then A and A^* have the same eigenvectors and their eigenvalues are conjugates of each other [86]. Then applying eigen decomposition on both matrices, we have $A = \Gamma\Lambda\Gamma^{-1}$ and $A^* = \Gamma\Lambda^*\Gamma^{-1}$, where Λ and Λ^* are, respectively, diagonal matrices of eigenvalues of A and A^* and Γ is the

matrix with the corresponding eigenvectors. Then, we have:

$$\begin{aligned} ||A||_2 &= \sqrt{\rho(A^*A)} = \sqrt{\rho(\Gamma\Lambda\Gamma^{-1}\Gamma\Lambda^*\Gamma^{-1})} \\ &= \sqrt{\rho(\Gamma\Lambda\Lambda^*\Gamma^{-1})} = \sqrt{\rho^2(A)} = \rho(A). \end{aligned} \tag{52}$$

The rest of proof directly follows from Theorem 1. □

We next define $\mathcal{DN}$ as the set of matrices diagonally similar to $\mathcal{N}$. In other words, for any matrix $A \in \mathcal{DN}$, there exists matrix $B \in \mathcal{N}$ and non-singular diagonal matrix W such that $WAW^{-1} = B$. Then, we can prove the result below.

Lemma 9. *If $A \in \mathcal{DN}$, A is stable under diagonal delays D_i if and only if $\rho(A) < 1$.*

Proof. Let $WAW^{-1} = B$, where B is normal and W is non-singular diagonal. Then, we have $\rho(A) = \rho(WAW^{-1}) = ||WAW^{-1}||_2 = ||A||_2$. According to Corollary 1, this implies that $\rho(A) < 1$ is both sufficient and necessary for A to be stable under delays D_i. □

The third class is $\mathcal{P}$, which consists of non-negative/non-positive matrices (i.e., $A \geq 0$ or $A \leq 0$). Combining the facts that $\rho(A) = \rho(|A|)$ and $\rho(A) \leq ||A||_s \leq \rho(|A|) = \rho(A)$ and invoking Corollary 2, we directly arrive at the following lemma.

Lemma 10. *If $A \in \mathcal{P}$, A is stable under diagonal delays D_i if and only if $\rho(A) < 1$.*

Similar to $\mathcal{DN}$, we define $\mathcal{DP}$ as the set of matrices diagonally similar to $\mathcal{P}$. Then, we have the following result.

Lemma 11. *If $A \in \mathcal{DP}$, A is stable under diagonal delays D_i if and only if $\rho(A) < 1$.*

Proof. Assume A is diagonally similar to a non-negative/non-positive matrix B. Then, $B = WAW^{-1}$ for some non-singular diagonal matrix W. Noticing that $\rho(|B|) = \rho(B) = \rho(A)$ and $\rho(A) \leq ||A||_s \leq \rho(|A|) = \rho(B) = \rho(A)$, we have

$\rho(A) = ||A||_s$, which directly follows from Corollary 2 that $\rho(A) < 1$ is both sufficient and necessary for system (31) to be stable. □

Next, define radial matrices $\mathcal{R}$ as the class of matrices satisfying $||A||_2 = \rho(A)$. Recalling that $||.||_2$ is induced from the L^2 norm and invoking Theorem 2, the following lemma is obvious.

Lemma 12. *If $A \in \mathcal{R}$, A is stable under diagonal delays D_i if and only if $\rho(A) < 1$.*

Analogous to Lemma 9, the last result also applies to $\mathcal{DR}$, which denotes any matrix diagonally similar to $\mathcal{R}$. Results obtained in this subsection are summarized as following.

Theorem 5. *The following matrices are stable under arbitrary diagonal delays D_i if and only if $\rho(A) < 1$: $\mathcal{N}$, $\mathcal{DN}$, $\mathcal{P}$, $\mathcal{DP}$, $\mathcal{R}$, and $\mathcal{DR}$.*

We conclude by identifying the relationship between different classes of matrices examined in this subsection.

Theorem 6. *The following relations hold: $\mathcal{N} \subset \mathcal{DN} \subset \mathcal{DR}$, $\mathcal{N} \subset \mathcal{R} \subset \mathcal{DR}$, and $\mathcal{P} \subset \mathcal{DP} \subset \mathcal{DR}$.*

Proof. We only present proofs of $\mathcal{DN} \subset \mathcal{DR}$ and $\mathcal{DP} \subset \mathcal{DR}$ and omit others for brevity. First let $A \in \mathcal{DN}$, then there exists non-singular diagonal matrix W such that $WAW^{-1} = B \in \mathcal{N}$. Since $\mathcal{N} \subset \mathcal{R}$ according to (52), we have $B \in \mathcal{R}$ and therefore $\mathcal{DN} \subset \mathcal{DR}$.

We next prove $\mathcal{DP} \subset \mathcal{DR}$. Since $A \in \mathcal{DP}$, there exists diagonal matrix W and $B \in \mathcal{P}$ such that $B = WAW^{-1}$. From (50), we know that for any $B \in \mathcal{P}$, there exists diagonal matrix V such that $||VBV^{-1}||_2 = \rho(B)$. Letting $C = UAU^{-1}$ and $U = WV$, we have $||C||_2 = \rho(B) = \rho(A) = \rho(C)$. This implies that A is diagonally similar to radial matrix C and therefore $\mathcal{DP} \subset \mathcal{DR}$. □

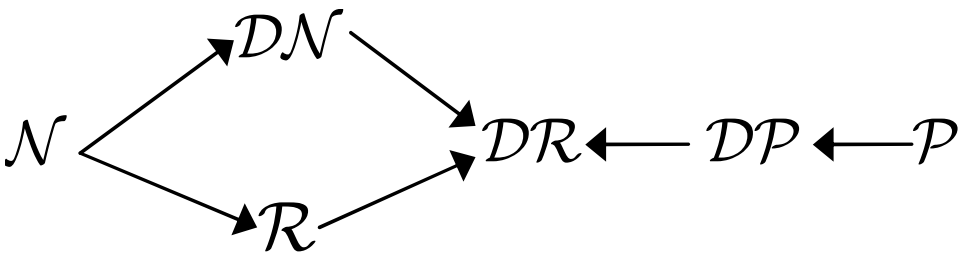


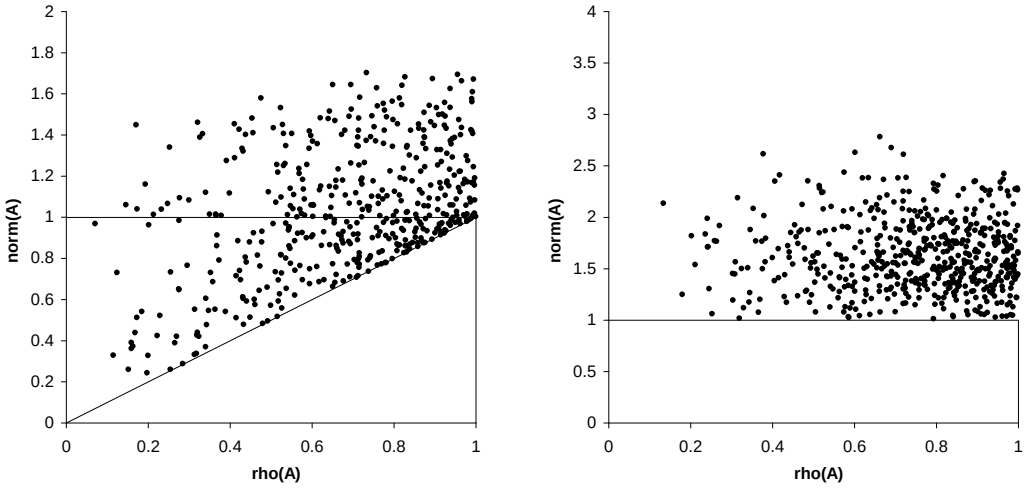
Fig. 11. Relationship between various classes of matrices

This result is also illustrated in Fig. 11, where notation $A \rightarrow B$ refers to $A \subset B$. As seen in the figure, $\mathcal{DR}$ is the widest class of matrices for which $\rho(A) < 1$ guarantees stability of (31). We leave exploration of more generic classes of delay-independent matrices for future work.

3.5 Discussion

In this subsection, we verify the obtained results using Matlab simulations. Our first step is to check sufficiency and also lack of necessity in the condition of Theorem 1. We generate 3000 two-by-two matrices and plot points (x, y) on a 2D plane, where $x = \rho(A)$ and $y = ||A||_2$. To detect instabilities, each matrix is tested with 100 random combinations of delay D_1 and D_2, each uniformly distributed in $[1, 30]$. We exclude all matrices with $\rho(A) > 1$ since these are a-priori known to be unstable. Out of 3000 random matrices, 1020 had $\rho(A) < 1$, out of which 468 were stable and 552 unstable under directional delay. Fig. 12(a) shows the stable points and (b) plots the unstable ones. From the first figure, notice that condition $||A||_2 \geq \rho(A)$ is never violated and Theorem 1 is *not* necessary for stability. At the same time, all unstable points in figure 12(b) are located above $||A||_2 = 1$, confirming the sufficiency of this condition.

Out of 468 stable matrices, 251 had $||A||_2 \geq 1$ and 331 had $||A||_\infty \geq 1$. Further-

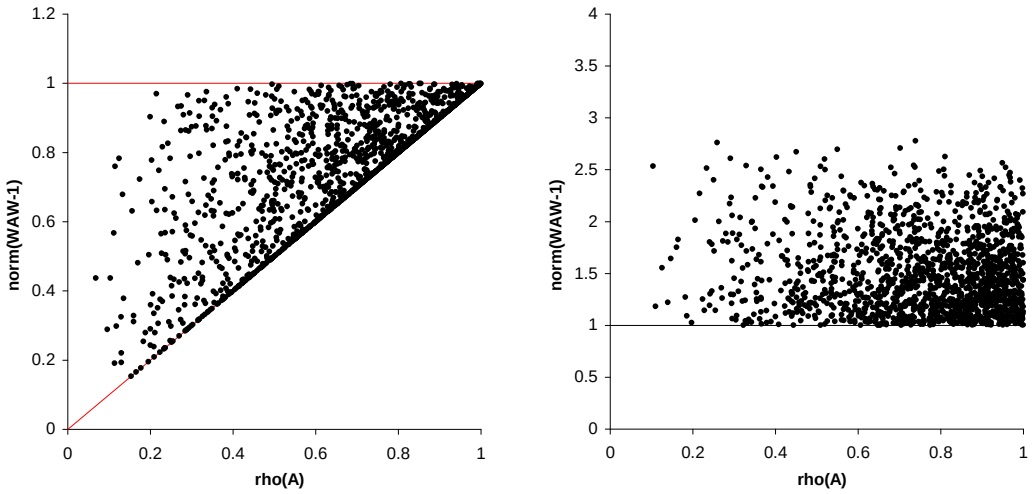


(a) stable (b) unstable

Fig. 12. Delayed stability as a function of $\rho(A)$ and $||A||_2$.

more, 240 matrices had both norms above 1 simultaneously and in 87% of the cases, $||A||_2$ was smaller than $||A||_\infty$. Out of 552 unstable matrices, all had $||A||_2 \geq 1$ and $||A||_\infty \geq 1$. Moreover, 86% of the cases had $||A||_2 < ||A||_\infty$. It thus appears that $||A||_2$ is a tighter norm in terms of obtaining the necessary and sufficient condition.

The next simulation generates 10000 random two-by-two matrices and examines whether condition $||A||_s = \inf_W ||WAW^{-1}||_2 < 1$ is in fact sufficient for stability of the delayed system. Fig. 13 plots 3535 stable/unstable points (1763 stable and 1772 unstable). The largest $||A||_s$ for a stable matrix was 0.9953 and the smallest for an unstable matrix was 1.0024. This demonstrates that for the *generated* matrices, condition $||A||_s < 1$ was both sufficient and necessary. We leave further investigation of necessity of this condition to future work. We next apply results obtained in the chapter to design practical congestion control protocols with delay-independent asymptotic stability.



(a) stable (b) unstable

Fig. 13. Delayed stability as a function of $\rho(A)$ and $\inf_W ||WAW^{-1}||_2$.

CHAPTER V

MAX-MIN KELLY CONTROL (MKC)

1 Classic Kelly Control

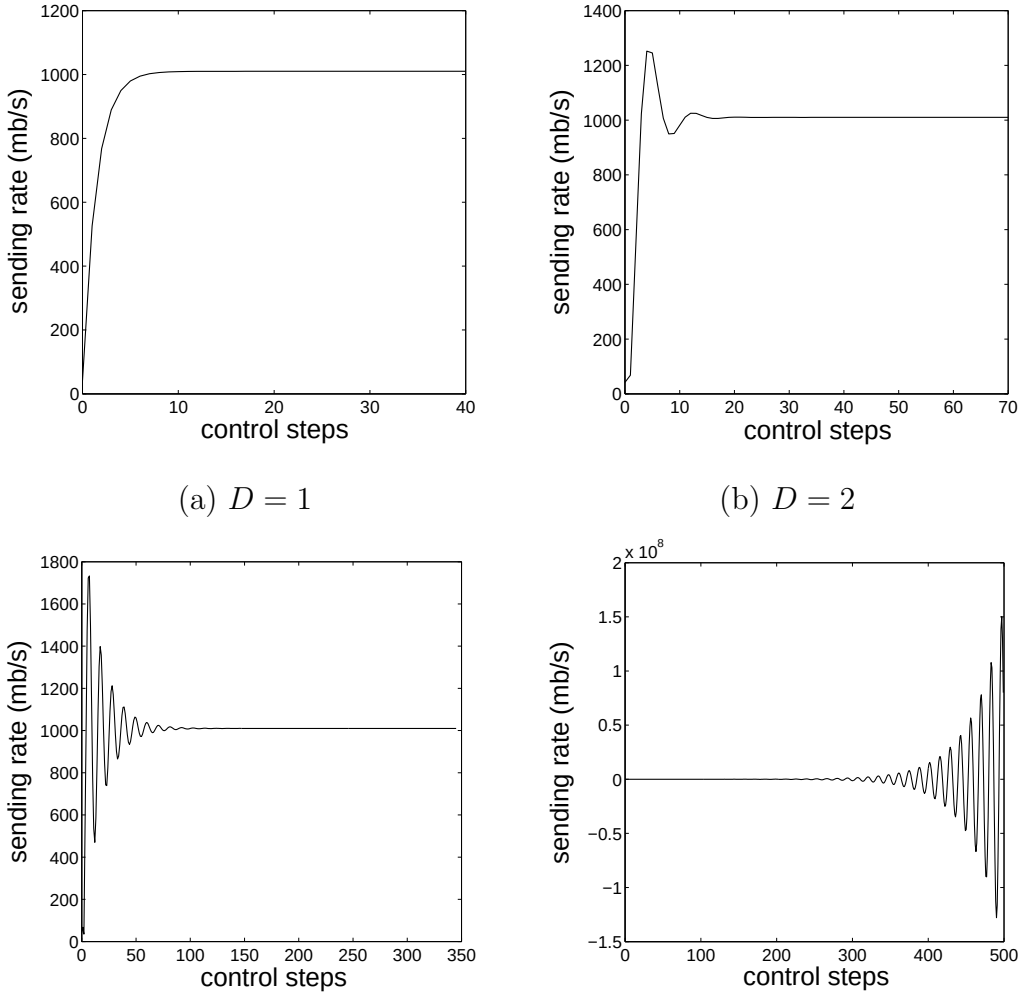
As described in Chapter II, modern Internet congestion control theory is pioneered by the framework proposed by Kelly [60], who applies microeconomics and optimization theory in modeling end-users' behavior in the distributed congestion control game. The proposed congestion control framework (1)-(2) is referred to as *Classic Kelly Control*. In this section, we discuss intuitive examples that explain the cryptic formulas (1)-(2) and demonstrate in simulation how delays affect stability of Kelly controls. We then show that the Classic Kelly control, or any mechanism that relies on the *sum* of feedback functions from individual routers, exhibits a tradeoff between linear convergence to efficiency and persistent stationary packet loss. We subsequently overcome both limitations in Section 2.

1.1 Delayed Stability Example

The following example illustrates stability problems of (1) when feedback delays are large. We assume a single-source, single-link configuration and utilize a congestion indication function that computes the estimated packet loss using instantaneous arrival rates:

$$p(n) = \frac{x(n) - C}{x(n)}, \tag{53}$$

where C is the link capacity and $x(n)$ is the flow rate at discrete step n.



(a) $D = 1$ (b) $D = 2$

(c) $D = 3$ (d) $D = 4$

Fig. 14. Stability of Kelly control under different feedback delays ($\kappa = 1/2$, $\omega = 10$ mb/s, and $C = 1,000$ mb/s).

We note that the price function $p(n)$ in the original Kelly control is nonnegative; however, as shown in [115], this results in slow linear AIMD-like probing for link capacity until the slowest link in the path is fully utilized, which is generally considered too slow for high-speed networks. Thus, under AQM feedback assumed throughout this dissertation, we allow *negative* values in (53), which signals the flow to increase its sending rate when $x(n) < C$. In section 3.1, we show that the negative component of packet-loss (53) improves convergence to efficiency from linear to exponential.

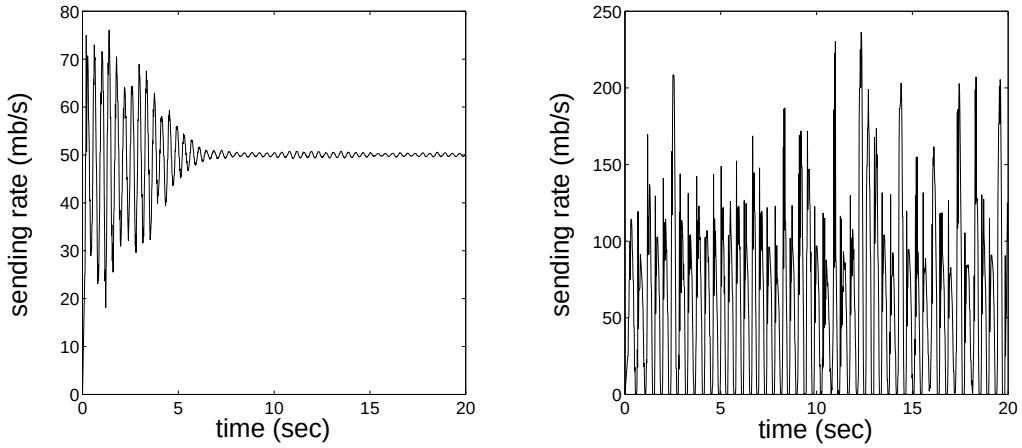
Applying (53) to Kelly control (1) yields a linear end-flow equation:

$$x(n) = x(n-1) + \kappa\omega - \kappa\big(x(n-D) - C\big). \tag{54}$$

Next, assume a particular set of parameters: $\kappa = 1/2$, $\omega = 10$ mb/s, and $C = 1,000$ mb/s. Solving the condition in (3), we have that the system is stable if and only if delay D is less than four time units. As illustrated in Figure 14(a), delay $D = 1$ keeps the system stable and monotonically convergent to its stationary point. Under larger delays $D = 2$ and $D = 3$ in Figures 14(b) and (c), the flow exhibits progressively increasing oscillations before entering the steady state. Eventually, as soon as D becomes equal to four time units, the system diverges as shown in Figure 14(d).

Using the same parameter κ and reducing ω to 20 kb/s, we examine (54) via `ns2` simulations, in which a single flow passes through a link of capacity 50 mb/s. We run the flow in two network configurations with the round-trip delay equal to 90 ms and 120 ms, respectively. As seen in Figure 15, the first flow reaches its steady state after decaying oscillations, while the second flow exhibits no convergence and periodically overshoots capacity C by 200%.

Since Kelly controls are unstable unless condition (3) is satisfied [51], a natural strategy to maintain stability is for each end-user i to adaptively adjusts its gain



(a) $D = 90$ ms (b) $D = 120$ ms

Fig. 15. Simulation results of the classic Kelly control under different delays ($\kappa = 1/2$, $\omega = 20$ kb/s, $C = 50$ mb/s).

parameter $\kappa_i \sim 1/D_i$ such that (3) is not violated. However, this method depends on reliable estimation of round-trip delays D_i and leads to unfairness between the flows with different RTTs.

1.2 Stationary Rate Allocation

As mentioned in the previous subsection, price function (53) should allow negative values, such that the convergence speed of Kelly control is improved from linear to exponential. However, we show next that this modification presents a problem in the stationary resource allocation. Consider a network of M resources and N homogeneous users (i.e., with the same parameters κ and ω). Further assume that resource j has capacity C_j, user i utilizes route r_i of length M_i (i.e., $M_i = |r_i|$), and packet-loss $\eta_i(n)$ fed back to user i is the *aggregate* feedback from all resources in path r_i. We further assume that there is no redundancy in the network (i.e., each user sends its packets through at least one resource and all resources are utilized by at least one user). Thus, we can define routing matrix $A_{N\times M}$ such that $A_{ij} = 1$ if

user i passes through resource j (i.e., $j \in r_i$) and $A_{ij} = 0$ otherwise. Further denote the j-th column of A by vector $\mathbf{V}_j$. Clearly, $\mathbf{V}_j$ identifies the set s_j of flows passing through router j.

Let $\mathbf{x}_j(n) = (x_1(n - D_{1j}^{\rightarrow}), x_2(n - D_{2j}^{\rightarrow}), \ldots, x_N(n - D_{Nj}^{\rightarrow}))$ be the vector of sending rates of individual users observed at router j at time instant n. In the spirit of (53), the packet loss of resource j at instant n can be expressed as:

$$p_j(n) = \frac{\mathbf{x}_j(n) \cdot \mathbf{V}_j - C_j}{\mathbf{x}_j(n) \cdot \mathbf{V}_j}, \tag{55}$$

where the dot operator represents vector multiplication. Then, we have the following result.

Lemma 13. *Let $\mathbf{x}^* = (x_1^*, x_2^*, \ldots, x_N^*)$ be the stationary rate allocation of Kelly control (1) with packet-loss function (55). Then $\mathbf{x}^*$ satisfies:*

$$\sum_{i=1}^{N} M_i x_i^* = \sum_{j=1}^{M} C_j + N\omega. \tag{56}$$

Proof. In the steady state, we can write the control equation of user i as following:

$$\begin{aligned} x_i^* &= (1 - \kappa\eta_i^*)x_i^* + \kappa\omega \\ &= \left(1 - \kappa \sum_{j \in r_i} \frac{\mathbf{x}^* \cdot \mathbf{V}_j - C_j}{\mathbf{x}^* \cdot \mathbf{V}_j}\right) x_i^* + \kappa\omega \ , \end{aligned} \tag{57}$$

where η_i^* denotes the stationary feedback seen by user i. Using simple manipulations in (57), we have:

$$M_i x_i^* - \sum_{j \in r_i} \left(\frac{x_i^* C_j}{\mathbf{x}^* \cdot \mathbf{V}_j}\right) = \omega \ . \tag{58}$$

Taking a summation of (58) for all N users, we get:

$$\sum_{i=1}^{N} M_i x_i^* = \sum_{i=1}^{N} \sum_{j \in r_i} \left(\frac{x_i^* C_j}{\mathbf{x}^* \cdot \mathbf{V}_j}\right) + N\omega \ . \tag{59}$$

Assuming no redundant users or resources, we can re-write (59) as follows:

$$\begin{aligned}\sum_{i=1}^{N} M_i x_i^* &= \sum_{j=1}^{M}\sum_{i\in s_j}\left(\frac{x_i^* C_j}{\mathbf{x}^*\cdot\mathbf{V}_j}\right) + N\omega \\ &= \sum_{j=1}^{M}(\mathbf{x}^*\cdot\mathbf{V}_j)\left(\frac{C_j}{\mathbf{x}^*\cdot\mathbf{V}_j}\right) + N\omega \\ &= \sum_{i=1}^{M} C_i + N\omega\ , \end{aligned} \tag{60}$$

which completes the proof. □

Lemma 13 provides a connection between the stationary resource allocation and the path length of each flow. Note that according to (56), the stationary rates x_i^* are constrained by the capacity of *all* resources instead of by that of individual bottlenecks. In fact, this observation shows an important difference between real network paths, which are limited by the *slowest* resource, and the model of proportional fairness augmented with (55), which takes into account the capacity of *all* resources in the network. As demonstrated in [115], this difference leads to significant overflow of slow routers and under-utilization of fast routers along a given path.

In the next section, we propose a new controller that overcomes both drawbacks of controller (1) (i.e., instability under delay and linear convergence to efficiency).

2 Stable Congestion Control

2.1 Max-min Kelly Control

We start our discussion with the following observations. First, we notice that in the classic Kelly control (1), the end-user decides its current rate $x_i(n)$ based on the most recent rate $x_i(n-1)$ and delayed feedback $\eta_j(n - D_{ij}^{\leftarrow})$. Since the latter carries information about $x_i(n - D_i)$, which was in effect *RTT time units earlier*, the

controller in (1) has no reason to involve $x_i(n-1)$ in its control loop. Thus, the sender quickly becomes unstable as the discrepancy between $x_i(n-1)$ and $x_i(n-D_i)$ increases. One natural remedy to this problem is to retard the reference rate to become $x_i(n-D_i)$ instead of $x_i(n-1)$ and allow the feedback to accurately reflect network conditions with respect to the first term of (1).

Second, to avoid unfairness[1] between flows, one must fix the control parameters of all end-users and establish a uniform set of equations that govern the system. Thus, we create a new notation in which $\kappa_i\omega_i = \alpha$, $\kappa_i = \beta$ and discretize Kelly control as following:

$$x_i(n) = x_i(n-D_i) + \alpha - \beta\eta_i(n)x_i(n-D_i), \tag{61}$$

where $\eta_i(n)$ is the congestion indication function of user i.

Next, to overcome the problems of proportional fairness described in the previous section and utilize negative network feedback, we combine (61) with max-min fairness (this idea is not new [56, 106]), under which the routers only feed back the packet loss of the *most-congested* resource instead of the combined packet loss of all links in the path:

$$\eta_i(n) = \max_{j\in r_i} p_j(n - D_{ij}^{\leftarrow}), \tag{62}$$

where $p_j(\cdot)$ is the congestion indication function of individual routers that depends only on the aggregate arrival rate $y_j(n)$ of end-users.

We call the resulting controller (61) *Max-min Kelly Control* (MKC) and emphasize that flows congested by the same bottleneck receive the *same* feedback and behave independently of the flows congested by the other links (see below for a justification of this). Therefore, we study in the next section the single-bottleneck case

[1]While "fairness" is surely a broad term, we assume its max-min version in this chapter.

since each MKC flow is always congested by only *one* router. Implementation details of how routers should feed back function (62) and how end-flows track the changes in the most-congested resource are presented in the simulation section.

2.2 Single-Link Stability of MKC

Consider an MKC system with a generic feedback function $\eta_i(n)$ in the form of (62), which we assume is differentiable in the stationary point and has the same first-order partial derivative for all end-users. Our goal is to derive sufficient and necessary conditions for the stability of (61)-(62) under arbitrarily delayed feedback.

We first prove MKC's stability in a single-link network containing N users $\{x_1, \ldots, x_N\}$ with corresponding delays to/from the bottleneck router given by $D_i^{\rightarrow}$ and $D_i^{\leftarrow}$. Then, we can simplify (61)-(62) by dropping index j of the bottleneck resource and expanding $\eta_i(n)$ in (61):

$$x_i(n) = x_i(n - D_i) + \alpha - \beta p(n - D_i^{\leftarrow}) x_i(n - D_i), \tag{63}$$

where

$$p(n) = p\Big(\sum_{u=1}^{N} x_u(n - D_u^{\rightarrow})\Big) \tag{64}$$

is the packet-loss function of the bottleneck router.

To invoke Theorem 5, our first step is to show stability of the following undelayed version of (63)-(64):

$$\begin{cases} x_i(n) &= \big(1 - \beta p(n-1)\big) x_i(n-1) + \alpha \\ p(n) &= p\Big(\sum_{u=1}^{N} x_u(n)\Big) \end{cases}. \tag{65}$$

Theorem 7. *Undelayed N-dimensional system (65) with feedback $p(n)$ that is common to all users has a symmetric Jacobian and is locally asymptotically stable if and*

only if:

$$0 < \beta p^* < 2, \tag{66}$$

$$0 < \beta p^* + \beta N x^* \left.\frac{\partial p}{\partial x_i}\right|_{\mathbf{x}^*} < 2, \tag{67}$$

where x^* is the fixed point of each individual user, vector $\mathbf{x}^* = (x^*, x^*, \ldots, x^*)$ is the fixed point of the entire system, and p^* is the steady-state packet loss.

Proof. We first derive the stationary point x^* of each individual user. Since all end-users receive the same feedback and activate the same response to it, all flows share the bottleneck resource fairly in the steady state, i.e., $x_i(n) = x^*$ for all i. Using simple manipulations in (65), we get the stationary individual rate x^* as following:

$$x^* = \frac{\alpha}{\beta p^*}. \tag{68}$$

Linearizing the system in $\mathbf{x}^*$:

$$\left.\frac{\partial f_i}{\partial x_i}\right|_{\mathbf{x}^*} = \left.\left(1 - \beta p - \beta x_i \frac{\partial p}{\partial x_i}\right)\right|_{\mathbf{x}^*}, \tag{69}$$

$$\left.\frac{\partial f_i}{\partial x_k}\right|_{\mathbf{x}^*} = \left.\left(-\beta x_i \frac{\partial p}{\partial x_k}\right)\right|_{\mathbf{x}^*}, \quad k \neq i, \tag{70}$$

where $f_i(\mathbf{x}) = (1 - \beta p(\mathbf{x}))x_i + \alpha$. Since packet loss depends on the *aggregate* rate of all users, $p(n)$ has the same first partial derivative evaluated in the fixed point for all users, which implies that for any users i and k, we have:

$$\left.\frac{\partial p}{\partial x_i}\right|_{\mathbf{x}^*} = \left.\frac{\partial p}{\partial x_k}\right|_{\mathbf{x}^*}. \tag{71}$$

This observation leads to a simple Jacobian matrix for MKC:

$$J = \begin{pmatrix} a & b & \cdots & b \\ b & a & \cdots & b \\ \vdots & \vdots & \ddots & \vdots \\ b & b & \cdots & a \end{pmatrix}, \tag{72}$$

where:

$$a = 1 - \beta p^* - \beta x^* \left.\frac{\partial p}{\partial x_i}\right|_{\mathbf{x}^*}, \quad b = -\beta x^* \left.\frac{\partial p}{\partial x_i}\right|_{\mathbf{x}^*}. \tag{73}$$

Clearly Jacobian matrix J is circulant[2] and thus its k-th eigenvalue λ_k is given by [16]:

$$\lambda_k = a + b(\zeta_k + \zeta_k^2 + \zeta_k^3 + \cdots + \zeta_k^{N-1}), \tag{74}$$

where $\zeta_k = e^{i2\pi k/N}$ $(k = 0, 1, \ldots, N-1)$ is one of the N-th roots of unity. We only consider the case of $N \geq 2$, otherwise the only eigenvalue is simply a. Then, it is not difficult to get the following result:

$$\lambda_k = \begin{cases} a + (N-1)b & \zeta_k = 1 \\ a + b\dfrac{\zeta_k - \zeta_k^N}{1 - \zeta_k} = a - b & \zeta_k \neq 1 \end{cases}, \tag{75}$$

where the last transition holds since $\zeta_k^N = 1$ for all k.

Next, recall that nonlinear system (65) is locally stable if and only if all eigenvalues of its Jacobian matrix J are within the unit circle [58]. Therefore, we get the following necessary and sufficient local stability conditions:

$$\begin{cases} |a - b| < 1 \\ |a + (N-1)b| < 1 \end{cases}. \tag{76}$$

[2]A matrix is called *circulant* if it is square and each of its rows can be obtained by shifting (with wrap-around) the previous row one column right [16].

To ensure that each λ_i lies in the unit circle, we examine the two conditions in (76) separately. First, notice that $|a-b| = |1-\beta p^*|$, which immediately leads to the following:

$$0 < \beta p^* < 2. \tag{77}$$

Applying the same substitution to the second inequality in (76), we obtain:

$$0 < \beta p^* + \beta N x^* \left.\frac{\partial p}{\partial x_i}\right|_{\mathbf{x}^*} < 2. \tag{78}$$

Thus, system (65) is locally stable if and only if both (77) and (78) are satisfied. □

According to the proof of Theorem 7, Jacobian J of the undelayed system (65) is symmetric and therefore is radial. Combining this property with Theorem 5, we arrive at the following result.

Corollary 3. *Heterogeneously delayed MKC (63)–(64) is locally asymptotically stable if and only if (66)-(67) are satisfied.*

Corollary 3 is a generic result that is applicable to MKC (61) with a wide class of congestion-indicator functions $\eta_i(n)$. Further note that for a given bottleneck resource with pricing function $p(n)$ and the set of is users, conditions (66)-(67) are easy to verify and do *not* depend on feedback delays, the number of hops in each path, or the routing matrix of all users. This is in contrast to many current studies [51, 77, 97, 99], whose results are dependent on individual feedback delays D_i and the topology of the network.

2.3 Stability of MKC with Heterogeneous α_i and β_i

In the last section, we established local stability of MKC, in which the proof holds only for a symmetric Jacobian matrix under the assumption of constant parameters α and β. We next utilize the techniques developed in Chapter IV to prove MKC's

stability under arbitrary parameters α_i and β_i. The resulting control equation thus becomes:

$$x_i(n) = x_i(n - D_i) + \alpha_i - \beta_i p(n) x_i(n - D_i), \tag{79}$$

where feedback $p(n)$ is a function of the combined incoming rate of all flows.

Theorem 8. *Assuming $p(n)$ is differentiable, (79) is stable under arbitrary delays D_i if:*

$$0 < \beta_i \Big(p^* + \sum_{i=1}^{N} x_i^* p' \Big) < 2, \quad i = 1, \ldots, N, \tag{80}$$

where x_i^* and p^* are, respectively, the equilibrium points of $x_i(n)$ and $p(n)$, and p' is the derivative of $p(n)$ evaluated in the equilibrium point.

Proof. Linearizing system (79) in the equilibrium point $\mathbf{x}^*$, we get the Jacobian matrix $A = (a_{ik})$ as follows:

$$a_{ik} = \begin{cases} -\beta_i x_i^* p' & i \neq k \\ 1 - \beta_i (p^* + x_i^* p') & i = k \end{cases}. \tag{81}$$

Introducing diagonal matrix $W = \mathrm{diag}(\sqrt{\beta_k x_k^*})$, we can construct a new matrix $B = (b_{ik}) = WAW^{-1}$ given below:

$$b_{ik} = \begin{cases} -\sqrt{\beta_i x_i^* \beta_k x_k^*} p' & i \neq k \\ 1 - \beta_i (p^* + x_i^* p') & i = k \end{cases}, \tag{82}$$

which is symmetric. Thus, matrix A is diagonally similar to a symmetric matrix B and, according to Lemma 9, is stable under diagonal delays D_i if and only if $\rho(A) < 1$.

We next define square matrix $C = (c_{ik})$ such that:

$$c_{ik} = \begin{cases} \beta_i x_i^* p' & i \neq k \\ \beta_i (p^* + x_i^* p') & i = k \end{cases}. \tag{83}$$

It is easy to see that $A = I - C$, where I is the identity matrix. Applying eigen decomposition on matrix C, we re-write C as $C = \Gamma\Lambda\Gamma^{-1}$, where Γ is a matrix of eigenvectors of C and Λ is a diagonal matrix with the corresponding eigenvalues. Then, we can compute the spectral radius $\rho(A)$ as follows:

$$\begin{aligned}\rho(A) &= \rho(I - \Gamma\Lambda\Gamma^{-1}) \\ &= \rho(\Gamma(I - \Lambda)\Gamma^{-1}) \\ &= \rho(I - \Lambda) \\ &= 1 - \rho(C). \end{aligned} \tag{84}$$

Then, condition $\rho(A) < 1$ leads to $0 < \rho(C) < 2$. Combining Property 2 and the assumption that $x_i^*, p^*, p' > 0$ [60, 115], we upper-bound $\rho(C)$ with $||C||_\infty$, which for $c_{ik} > 0$ leads to $\rho(C) \leq \max_i(\sum_{k=1}^N c_{ik})$. This immediately yields (80). □

It is easy to see that by letting $\alpha = \alpha_i$ and $\beta = \beta_i$ for all i, Theorem 8 directly translates to the sufficient condition of [115, Theorem 3].

2.4 Exponential MKC

To understand the practical implications of MKC, we next associate MKC with the following packet-loss function $p(n)$ in (64):

$$p(n) = \frac{\sum_{u=1}^N x_u(n - D_u^{\rightarrow}) - C}{\sum_{u=1}^N x_u(n - D_u^{\rightarrow})}, \tag{85}$$

where we again assume a network with a single link of capacity C and N users. This is a rather standard packet-loss function with the exception that we allow it to become negative when the link is under-utilized. As we show in the next section, (85) achieves exponential convergence to efficiency, which explains why we call the combination of (63),(85) *Exponential MKC* (EMKC).

Theorem 9. *Heterogeneously delayed single-link EMKC (63),(85) is locally asymptotically stable if and only if* $0 < \beta < 2$.

Proof. We first derive the fixed point of EMKC. Notice that in the proof of Theorem 7, we established the existence of a unique stationary point $x_i^* = x^*$ for each flow. Then assuming EMKC packet-loss function (85), we have:

$$p^* = \frac{Nx^* - C}{Nx^*}. \tag{86}$$

Combining (86) and (68), we get the stationary point x^* of each end-user:

$$x^* = \frac{C}{N} + \frac{\alpha}{\beta}. \tag{87}$$

Denoting by $X(n) = \sum_{i=1}^{N} x_i(n)$ the combined rate of all N end-users at time n, the corresponding combined stationary rate X^* is:

$$X^* = Nx^* = C + N\frac{\alpha}{\beta}. \tag{88}$$

Next, recall from Theorem 7 that stability conditions (66)-(67) must hold for the delayed system to be stable. Consequently, we substitute pricing function (85) into (67) and obtain with the help of (88):

$$\beta p^* + \beta N x^* \left.\frac{\partial p(n)}{\partial x(n)}\right|_{\mathbf{x}^*} = \beta p^* + \frac{\beta N x^* C}{N^2 x^{*2}} = \beta. \tag{89}$$

Thus, condition (67) becomes

$$0 < \beta < 2. \tag{90}$$

Notice that in the steady state, packet loss probability p^* is no larger than one. Hence, the last condition is more conservative than (66), which allows us to conclude that when $0 < \beta < 2$, all eigenvalues of Jacobian matrix J are inside the unit circle.

Applying Corollary 3, heterogeneously delayed EMKC defined by (63) and (85) is also locally asymptotically stable if and only if $0 < \beta < 2$. □

We next prove global asymptotic stability of EMKC under homogeneous delay.

2.5 Global Stability of EMKC under Constant Delay

Recall that global asymptotic stability of a nonlinear dynamic system requires both Lyapunov stability and global quasi-asymptotic stability (whose definition follows later) in the *unique* stable fixed point [39]. Note that we proved local asymptotic stability of EMKC in the preceding section, which implies Lyapunov stability of the system. Thus, our remaining task is to prove that EMKC will converge to the unique fixed point regardless of its initial conditions. To accomplish this, we first consider several auxiliary results.

2.5.1 Preliminaries

We start with a very simple lemma.

Lemma 14. *For an arbitrary sequence v_n such that $v_n \to 0$ for $n \to \infty$ and another sequence α_n such that $\forall n > n_0$: $|\alpha_n| < 1 - \varepsilon$, where $\varepsilon > 0$, the following recurrence converges to zero regardless of the value of x_0: $x_n = \alpha_n x_{n-1} + v_n$.*

Proof. Defining a new set of variables such that $y_n = x_{n+n_0}$, $\beta_n = \alpha_{n+n_0}$, and $u_n = v_{n+n_0}$ to shift recurrence x_n by n_0 time units forward and skipping the transient region of the evolution of x_n when α_n can potentially be larger than 1, we obtain $y_n = \beta_n y_{n-1} + u_n$. Using these assignments, $|\beta_n|$ is less than $1 - \varepsilon$ for all $n \geq 0$. We next demonstrate that sequence y_n converges to zero, which implies that x_n does too.

Recursively expanding y_n, for $n \geq 2$, we get:

$$y_n = \prod_{i=1}^{n} \beta_i y_0 + u_n + \sum_{i=1}^{n-1} u_i \prod_{j=i+1}^{n} \beta_j. \tag{91}$$

For convenience of presentation, let

$$S_1(n) = \prod_{i=1}^{n} \beta_i y_0 + u_n \text{ and } S_2(n) = \sum_{i=1}^{n-1} \Big(u_i \prod_{j=i+1}^{n} \beta_j \Big). \tag{92}$$

Since $|\beta_n| < 1 - \varepsilon$ and u_n is a time-shifted version of v_n, we immediately obtain that $S_1(n) \to 0$ as $n \to \infty$. Next examine $S_2(n)$ and show that it also tends to zero for large n. Re-writing (92):

$$|S_2(n)| \leq \sum_{i=1}^{n-1} \Big(|u_i| \prod_{j=i+1}^{n} |\beta_j| \Big). \tag{93}$$

Again since $|\beta_n| < 1 - \varepsilon$, we have:

$$|S_2(n)| \leq \sum_{i=1}^{n-1} |u_i| (1-\varepsilon)^{n-i} = G_1(n) + G_2(n), \tag{94}$$

where we define:

$$G_1(n) = \sum_{i=1}^{n/2} |u_i| (1-\varepsilon)^{n-i} \text{ and } G_2(n) = \sum_{i=n/2+1}^{n-1} |u_i| (1-\varepsilon)^{n-i}. \tag{95}$$

To show that both $G_1(n)$ and $G_2(n)$ converge to zero, we need the following notations: $m_1(n) = \max(|u_1|, \ldots, |u_{n/2}|)$ and $m_2(n) = \max(|u_{n/2+1}|, \ldots, |u_{n-1}|)$. Then we have:

$$\begin{aligned} G_1(n) &\leq m_1(n) \sum_{i=1}^{n/2} (1-\varepsilon)^{n-i} = m_1(n) \sum_{j=n/2}^{n-1} (1-\varepsilon)^j \\ &= m_1(n) \left(\sum_{j=0}^{n-1} (1-\varepsilon)^j - \sum_{j=0}^{n/2-1} (1-\varepsilon)^j \right) \\ &= m_1(n) \frac{(1-\varepsilon)^n - (1-\varepsilon)^{n/2}}{\varepsilon}. \end{aligned} \tag{96}$$

Since $m_1(n)$ is bounded and $0 < \varepsilon < 1$, $G_1(n) \to 0$. For $G_2(n)$, we have:

$$G_2(n) \leq m_2(n) \sum_{i=n/2+1}^{n-1} (1-\varepsilon)^{n-i} \leq m_2(n) \sum_{i=0}^{\infty} (1-\varepsilon)^i = \frac{m_2(n)}{\varepsilon}. \tag{97}$$

Notice that since both $u_{n/2}$ and u_n converge to zero, then so must $m_2(n)$. Therefore, we get $G_2(n) \to 0$, which leads to $S_2(n) \to 0$ and hence $y_n \to 0$. □

We next present our main result of this section.

Theorem 10. *Assume a nonlinear system $x_n = f(x_{n-1}, y_{n-1})$, where function $f(x, y)$ is linear in both arguments, i.e., $f(x, y) = a + bx + cy + dxy$, for some constants $a - d$. Further assume that y_n converges to a stationary point y^* as $n \to \infty$ and form another system, which replaces y_n with y^* in system x_n: $\tilde{x}_n = f(\tilde{x}_{n-1}, y^*)$. Then, system x_n converges if and only if system $\tilde{x}_n$ converges, in which case the two stationary points are the same regardless of the initial points x_0 and $\tilde{x}_0$ in which each system is started:* $\lim_{n\to\infty} |x_n - \tilde{x}_n| = 0$.

Proof. We again only prove the sufficient condition. The necessary condition follows by reversing the order of steps. First notice that system $\tilde{x}_n$ is stable (bounded) if and only if $|b + dy^*| < 1$. Next denote by Δx_n the absolute distance between the trajectories of the two systems at time n: $\Delta x_n = x_n - \tilde{x}_n$. Further let $\Delta y_n = y_n - y^*$ be the distance of y_n from its stationary point. Then we can write:

$$\begin{aligned} \Delta x_{n+1} &= x_{n+1} - \tilde{x}_{n+1} = f(x_n, y_n) - f(\tilde{x}_n, y^*) \\ &= f(x_n, y_n) - f(\tilde{x}_n, y_n) + f(\tilde{x}_n, y_n) - f(\tilde{x}_n, y^*) \\ &= (b + dy_n)\Delta x_n + (c + d\tilde{x}_n)\Delta y_n. \end{aligned} \tag{98}$$

Next notice that (98) defines a recursive relationship on Δx_n:

$$\Delta x_n = \alpha_n \Delta x_{n-1} + v_n, \tag{99}$$

where $\alpha_n = b + dy_n$ and $v_n = (c + d\tilde{x}_n)\Delta y_n$. First, since $\tilde{x}_n$ is bounded and $\Delta y_n \to 0$ as $n \to \infty$, we have $v_n \to 0$ for large n. Second, since $|b + dy^*| < 1$, there exists such ε that: $|b + dy^*| < 1 - 2\varepsilon$.

Since $y_n \to y^*$, there exists such n_0 that $\forall n > n_0$, sequence α_n is bounded by the following:

$$|\alpha_n| = |b + dy_n| < 1 - \varepsilon, \forall n > n_0. \tag{100}$$

Thus, system (99) satisfies the conditions of Lemma 14 and converges to zero as $n \to \infty$. □

2.5.2 Main Results

We next show global stability of the combined rate $X(n)$ of N EMKC flows sharing a single bottleneck and convergence of loss $p(n)$ to p^* regardless of the behavior of flow rates $x_i(n)$.

Lemma 15. *When $0 < \beta < 2$, the combined rate $X(n)$ of EMKC is globally asymptotically stable under constant delay and converges to $X^* = C + N\alpha/\beta$ at an exponential rate.*

Proof. Assume that delay D is constant. Combining (61)-(62) and taking the summation for all N flows, we get that EMKC's combined rate $X(n) = \sum_i x_i(n)$ forms a linear system:

$$X(n) = \left(1 - \beta\frac{X(n-D) - C}{X(n-D)}\right) X(n-D) + N\alpha = (1-\beta)X(n-D) + \beta C + N\alpha. \tag{101}$$

It is clear that the above linear system is stable if and only if $0 < \beta < 2$. Since convergence of linear systems implies global asymptotic stability, we can conclude that $X(n)$ is globally stable regardless of individual flow trajectories $x_i(n)$.

We next show the convergence speed of $X(n)$. Recursively expanding (101), we

have:

$$X(n) = (1-\beta)^{\frac{n}{D}}(X_0 - X^*) + X^*, \tag{102}$$

where X_0 is the combined initial rate and $X^* = C + N\alpha/\beta$ is the combined stationary rate of all flows. Notice that for $0 < \beta < 2$, the first term in the above equation approaches zero exponentially fast and $X(n)$ indeed converges to X^*. □

Using (62), it is not difficult to see that $p(n)$ can be expressed as $p(n) = 1 - C/X(n)$. Combining this observation with the result of Lemma 15, we immediately have the following corollary.

Corollary 4. *When $0 < \beta < 2$, EMKC's packet loss $p(n)$ converges to $p^* = N\alpha/(C\beta + N\alpha)$ regardless of the initial rates of the flows or their individual rates $x_i(n)$.*

Before showing global stability of EMKC, we first review the following stability concept that describes asymptotic properties of a dynamic system.

Definition 5. *[39] A point $\mathbf{x}^*$ is globally quasi-asymptotically stable if and only if for all $\varepsilon > 0$ there exists n_0 such that for all $n > n_0 : |\mathbf{x}(n) - \mathbf{x}^*| < \varepsilon$ regardless of the initial point $\mathbf{x}(0)$.*

According to Corollary 3, EMKC is *locally* quasi-asymptotically stable in its unique fixed point $\mathbf{x}^*$. In what follows, we prove that each individual flow rate $x_i(n)$ is globally quasi-asymptotically stable, which implies that the entire system of flows $\mathbf{x}(n) = \langle x_1(n), \ldots, x_N(n)\rangle$ also exhibits global quasi-asymptotic stability.

Theorem 11. *Assuming an N-flow EMKC system with constant delay D and an arbitrary initial point $\mathbf{x}(0) = \langle x_1(0), \ldots, x_N(0)\rangle$, $x_i(n)$ converges to $x^* = C/N + \alpha/\beta$ if and only if $0 < \beta < 2$.*

Proof. We start with the sufficient condition. Under constant delay D, each EMKC flow activates a rate adjustment every D time units. Thus, we can define a new set of flows $\{u_i(t)\}$, which operate in time units scaled by a factor of D. Under this notation, we can write $x_i(n) = u_i(n/D) = u_i(t)$ and $x_i(n-D) = u_i(n/D-1) = u_i(t-1)$. Notice that $u_i(t)$ has the same exact stability properties as $x_i(n)$. Select an arbitrary flow u_i and focus on its stability: $u_i(t) = f(u_i(t-1), p(t-1))$, where $p(t)$ is the packet loss at time t and $f(x,y)$ is given by:

$$f(x,y) = (1-\beta y)x + \alpha. \tag{103}$$

Then form a new system $\tilde{u}(t) = f(\tilde{u}(t-1), p^*) = (1-\beta p^*)\tilde{u}(t-1) + \alpha$, where $\tilde{u}(0) = u_i(0)$, and notice that the solution to this recurrence is stable if and only if $|b + dy^*| = |1 - \beta p^*| < 1$. This condition is automatically satisfied using the proof of EMKC's local stability in Theorem 9. According to Corollary 4, we notice that $p(t)$ converges to its unique stationary point p^* regardless of $\mathbf{x}(0)$. Since (103) is linear in each argument, we can apply Theorem 10 and immediately obtain that $u_i(n) \to \tilde{u}^* = C/N + \alpha/\beta$ and is therefore quasi-asymptotically stable regardless of the initial points $u_i(0)$ or $\mathbf{x}(0)$. Repeating the same argument for all flows i, we establish their individual convergence.

The necessity of condition $0 < \beta < 2$ directly follows from Theorem 9. □

Combining EMKC's Lyapunov and global quasi-asymptotic stability, we have:

Corollary 5. *EMKC is globally asymptotically stable under constant feedback delay D if and only if $0 < \beta < 2$.*

3 Performance of EMKC

3.1 Convergence to Efficiency

In this section, we show that EMKC converges to efficiency exponentially fast.

Lemma 16. *For $0 < \beta < 2$ and constant delay D, the combined rate $X(n)$ of EMKC is globally asymptotically stable and converges to $X^* = C + N\alpha/\beta$ at an exponential rate.*

Proof. Since delays do not affect stability of EMKC, assume a constant feedback delay D and re-write (63):

$$x_i(n) = (1 - \beta p(n - D))x_i(n - D) + \alpha, \tag{104}$$

where $p(n)$ is the undelayed version of (85). Taking the summation of (104) for all N flows, we get that EMKC's combined rate $X(n) = \sum_{i=1}^{N} x_i(n)$ forms a linear system:

$$\begin{aligned} X(n) &= \left(1 - \beta\frac{X(n - D) - C}{X(n - D)}\right) X(n - D) + N\alpha \\ &= (1 - \beta)X(n - D) + \beta C + N\alpha. \end{aligned} \tag{105}$$

It is clear that the above linear system is stable if and only if $0 < \beta < 2$. Since convergence of linear systems implies global asymptotic stability, we conclude that $X(n)$ is globally stable regardless of individual flow trajectories $x_i(n)$.

We next show the convergence speed of $X(n)$. Recursively expanding the last equation, we have:

$$X(n) = (1 - \beta)^{\frac{n}{D}}(X_0 - X^*) + X^*, \tag{106}$$

where X_0 is the initial combined rate of all flows and $X^* = C + N\alpha/\beta$ is the combined stationary rate. Notice that for $0 < \beta < 2$, the first term in (106) approaches zero exponentially fast and $X(n)$ indeed converges to X^*. □

From (106), notice that the value of β affects the convergence behavior of EMKC. Specifically, for $0 < \beta \leq 1$, the system monotonically converges to the stationary point; however, for $1 < \beta < 2$, the system experiences decaying oscillations before reaching the stationary point, which are caused by the oscillating term $(1-\beta)^{n/D}$ in (106). Thus, in practical settings, β should be chosen in the interval $(0, 1]$, where values closer to 1 result in faster convergence to efficiency.

3.2 Convergence to Fairness

We next investigate the convergence rate of EMKC to fairness. To better understand how many steps EMKC requires to reach a certain level of max-min fairness, we utilize a simple metric that we call *ε-fairness*. For a given small positive constant ε, a rate allocation $(x_1, x_2, \ldots, x_N)$ is ε-fair, if:

$$f = \frac{\min_{i=1}^{N} x_i}{\max_{j=1}^{N} x_j} \geq 1 - \varepsilon. \tag{107}$$

Generally, ε-fairness assesses max-min fairness by measuring the worst-case ratio between the rates of any pair of flows. Given the definition in (107), we have the following result.

Theorem 12. *Consider an EMKC network with N users and a bottleneck link of capacity C. Assuming that the system is started in the maximally unfair state, ε-fairness is reached in θ_M steps, where:*

$$\theta_M = \frac{(C + N\frac{\alpha}{\beta})\big(\log N - \log \varepsilon\big)}{N\alpha} + \Theta\Big(\frac{N\alpha}{C}\Big). \tag{108}$$

Proof. Let (x, y) be the pair of initially maximally unfair flows, i.e., the difference between their initial sending rates $\Delta(0) = y(0) - x(0)$, where $y(0) > x(0)$, is maximal among that of any two flows. Notice that under MKC and the assumption of syn-

chronization, (x, y) are always maximally unfair during the entire process till certain fairness threshold is reached. Then we have:

$$\begin{aligned}\Delta(i) &= y(i) - x(i) \\ &\approx \big(y(0) - x(0)\big)\left(1 - \frac{N\alpha}{X^*}\right)^i \\ &= \Delta(0)\left(1 - \frac{N\alpha}{X^*}\right)^i . \end{aligned} \tag{109}$$

Thus, the fairness index at step i becomes:

$$\begin{aligned} f(i) &= \frac{x(i)}{y(i)} = \frac{y(i) - \Delta(i)}{y(i)} = 1 - \frac{\Delta(0)\left(1 - \frac{N\alpha}{X^*}\right)^i}{y(i)} \\ &\geq 1 - \frac{\Delta(0)\left(1 - \frac{N\alpha}{X^*}\right)^i}{y^*}, \end{aligned} \tag{110}$$

since $y(i) \geq y^*$. Hence, to achieve ε-fairness, we have:

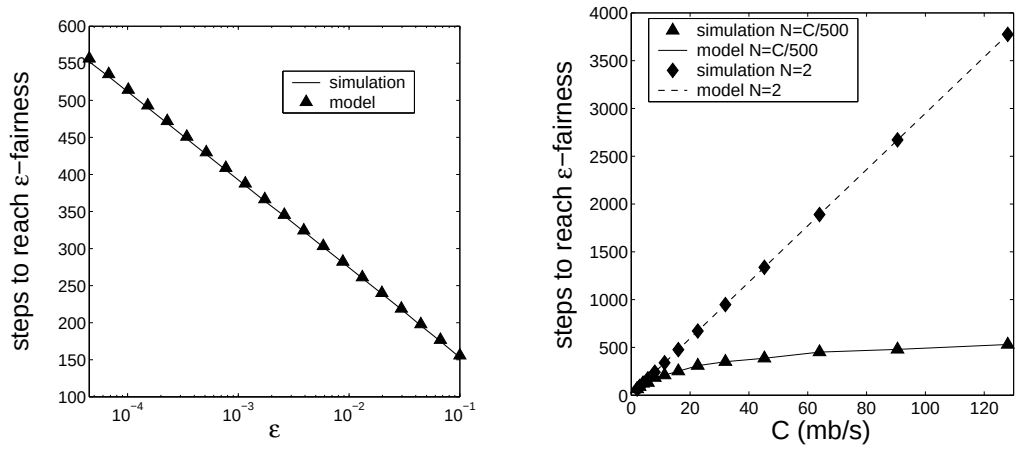
$$f(n) \geq 1 - \frac{\Delta(0)\left(1 - \frac{N\alpha}{X^*}\right)^n}{y^*} \geq 1 - \varepsilon \ , \tag{111}$$

which yields:

$$\begin{aligned}\theta_M &\leq \log_{1-\frac{N\alpha}{X^*}} \frac{y^*\varepsilon}{\Delta(0)} = \frac{\log\big(y^*\varepsilon/\Delta(0)\big)}{\log(1 - \frac{N\alpha}{X^*})} \\ &\approx -\frac{X^* \log(y^*\varepsilon/\Delta_0)}{N\alpha} + \Theta\Big(\frac{N\alpha}{C}\Big). \end{aligned} \tag{112}$$

Assuming $N\alpha/C \ll 1$ and substituting $y^* = C/N + \alpha/\beta$ and $X^* = C + N\alpha/\beta$ in (112), we get:

$$\begin{aligned}\theta_M &\leq -\frac{(C + N\frac{\alpha}{\beta})\big(\log(\frac{C}{N} + \frac{\alpha}{\beta}) + \log\varepsilon - \log\Delta(0)\big)}{N\alpha} \\ &\approx \frac{(C + N\frac{\alpha}{\beta})\big(\log N + \log\Delta(0) - \log C - \log\varepsilon\big)}{N\alpha} \\ &= \frac{(C + N\frac{\alpha}{\beta})\big(\log N - \log\varepsilon\big)}{N\alpha}. \end{aligned} \tag{113}$$



(a) (b)

Fig. 16. (a) Verification of model (108) against EMKC simulations ($C = 1$ mb/s, $\alpha = 10$ kb/s, and $\beta = 0.5$). (b) Exponential and linear rates of convergence to fairness for EMKC ($\varepsilon = 0.1$).

Adding the omitted terms on the order of $\Theta(N\alpha/C)$ to (113), we arrive at (108). □

A comparison of model (108) to simulation results is shown in Figure 16(a) (note that in the figure, the model is drawn as a solid line and simulation results are plotted as isolated triangles). In this example, we use a bottleneck link of capacity $C = 1$ mb/s shared by two EMKC flows, which are initially separated by the maximum distance, i.e., $x_1(0) = 0, x_2(0) = C$. As seen from the figure, the number of steps predicted by (108) agrees with simulation results for a wide range of ε.

As noted in the previous section, parameter β is responsible for the convergence speed to efficiency; however, as seen in (108), it has little effect on the convergence rate to fairness (since typically $N\alpha \ll C$). In contrast, parameter α has no effect on convergence to efficiency in (106), but instead determines the convergence rate to fairness in the denominator of (108). Also observe the following interesting fact about (108) and the suitability of EMKC for high-speed networks. As C increases, the behavior of θ_M changes depending on whether N remains fixed or not. For a

constant N, (108) scales linearly with C; however, if the network provider increases the number of flows as a function of C and keeps $N = \Theta(C)$, ε-fairness is reached in $\Theta(\log C)$ steps. This implies exponential convergence to fairness and very good scaling properties of EMKC in future high-speed networks. Both types of convergence are demonstrated in Figure 16(b) for constant $N = 2$ and variable $N = \lceil C/500 \rceil$ (for the latter case, C is taken to be in kb/s). As the figure shows, both linear and logarithmic models obtained from (108) match simulations well.

We next compare EMKC's convergence speed to that of rate-based AIMD. Recall that rate-based AIMD(α, β) adjusts its sending rate according to the following rules assuming $\alpha > 0$ and $0 < \beta < 1$:

$$x(t) = \begin{cases} x(t - RTT) + \alpha & \text{per RTT} \\ (1 - \beta)x(t - RTT) & \text{per loss} \end{cases}. \tag{114}$$

Theorem 13. *Under the assumptions of Theorem 12, rate-based AIMD reaches ε-fairness in θ_A steps, where:*

$$\theta_A = \frac{(C + N\frac{\alpha}{\beta})\big(\log N - \log \varepsilon\big)}{-N\alpha \log(1 - \beta)/\beta} + \Theta\Big(\frac{N\alpha}{C}\Big). \tag{115}$$

Proof. Assume that flows are synchronized and reach full link utilization at time instants $\tau_1, \tau_2, \ldots$ We again assume that $N\alpha \ll C$ and neglect the random amount of overshoot, which generally fluctuates between 0 and $N\alpha$. Analysis below focuses on two maximally unfair flows x and y (i.e., $x(0) = 0, y(0) = C$) since these flows solely determine max-min fairness of the system. After packet loss is detected at time τ_j, the immediate rate reduction brings rates $x(\tau_j)$ and $y(\tau_j)$ to $(1-\beta)x(\tau_j)$ and $(1 - \beta)y(\tau_j)$ and the combined rate of all users drops to $(1 - \beta)C$. Following this reduction, the combined rate is then incremented by $N\alpha$ per RTT until it reaches

C at time τ_{j+1}. This implies that at the end of interval $[\tau_j, \tau_{j+1}]$, each flow's rate is increased by $w = \beta C/N$, meaning that flows x and y climb back to $(1-\beta)x(\tau_j) + w$ and $(1-\beta)y(\tau_j) + w$, respectively. Hence, the new rates when the flows hit the efficiency line for the j-th time are:

$$x(\tau_j) = (1-\beta)x(\tau_{j-1}) + \frac{\beta C}{N}, \tag{116}$$

$$y(\tau_j) = (1-\beta)y(\tau_{j-1}) + \frac{\beta C}{N}. \tag{117}$$

It is not difficult to see that the distance between any two flows shrinks exponentially:

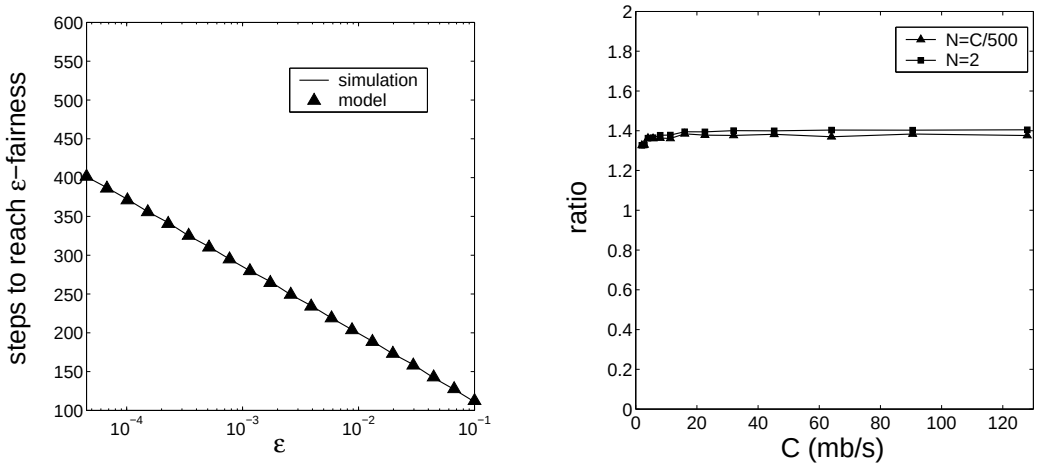
$$\begin{aligned} \Delta(\tau_j) &= y(\tau_j) - x(\tau_j) = (1-\beta)\big(y(\tau_{j-1}) - x(\tau_{j-1})\big) \\ &= (1-\beta)\Delta(\tau_{j-1}) = \Delta(0)(1-\beta)^j. \end{aligned} \tag{118}$$

Using simple manipulations, we have max-min fairness:

$$\begin{aligned} f(\tau_j) &= \frac{x(\tau_j)}{y(\tau_j)} = \frac{y(\tau_j) - \Delta(\tau_j)}{y(\tau_j)} = 1 - \frac{\Delta(\tau_j)}{y(\tau_j)} \\ &\geq 1 - \frac{(1-\beta)^j \Delta(0)}{C/N} = 1 - q(1-\beta)^j, \end{aligned} \tag{119}$$

where constant $q = N\Delta(0)/C$.

The number of packet-loss intervals to reach ε-fairness is no more than $\log_{1-\beta}(\varepsilon/q)$, while the number of increase steps during each packet-loss interval is $\beta C/(N\alpha)$. Thus,



(a) (b)

Fig. 17. (a) Verification of model (115) against AIMD simulations ($C = 1$ mb/s, $\alpha = 10$ kb/s, and $\beta = 0.5$). (b) Ratio θ_M/θ_A for fixed and variable N.

the total number of steps to convergence is:

$$\begin{aligned}
\theta_A &= \left(\left\lceil \frac{\beta C}{N\alpha} \right\rceil\right) \log_{1-\beta} \frac{\varepsilon}{q} \\
&\approx \left(\frac{\beta C}{N\alpha} + 1\right) \log_{1-\beta} \frac{C\varepsilon}{N\Delta(0)} \\
&= \frac{(C + N\alpha/\beta)\big(\log N + \log \Delta(0) - \log C - \log \varepsilon\big)}{-N\alpha \log(1-\beta)/\beta} \\
&= \frac{(C + N\frac{\alpha}{\beta})\big(\log N - \log \varepsilon\big)}{-N\alpha \log(1-\beta)/\beta}.
\end{aligned} \tag{120}$$

Accounting for random overshoot and neglected terms, we get (115). □

Figure 17(a) verifies that model (115) is also very accurate for a range of different ε. Notice from (108) and (115) that the speed of convergence to fairness between AIMD and EMKC differs by a certain constant coefficient. The following corollary formalizes this observation.

Corollary 6. *For the same parameters N, α, β such that $N\alpha \ll C$, AIMD reaches ε-fairness $\theta_M/\theta_A = -\log(1-\beta)/\beta$ times faster than EMKC.*

For TCP and $\beta = 0.5$, this difference is by a factor of $2 \log 2 \approx 1.39$, which holds regardless of whether N is fixed or not as demonstrated in Figure 17(b). We should finally note that as term $\Theta(N\alpha/C)$ becomes large, MKC's performance improves and converges to that of AIMD.

3.3 Packet Loss

As seen in previous sections, EMKC converges to the combined stationary point $X^* = C + N\alpha/\beta$, which is above capacity C. This leads to constant (albeit usually small) packet loss in the steady state. However, the advantage of this framework is that EMKC does not oscillate or react to individual packet losses, but instead adjusts its rate in response to a *gradual* increase in $p(n)$. Thus, a small amount of FEC can provide a smooth channel to fluctuation-sensitive applications such as video telephony and various types of real-time streaming. Besides being a stable framework, EMKC is also expected to work well in wireless networks where congestion-unrelated losses will not cause sudden reductions in the flow rates.

Also notice that EMKC's steady-state packet loss $p^* = N\alpha/(C\beta + N\alpha)$ increases linearly with the number of competing flows, which causes problems in scalability to a large number of flows. However, it still outperforms AIMD, whose increase in packet loss is quadratic as a function of N [72]. Furthermore, if the network provider keeps $N = \Theta(C)$, EMKC achieves *constant* packet loss in addition to exponential convergence to fairness.

Finally, observe that if the router is able to count the number of flows, zero packet loss can be obtained by adding a constant $\Delta = N\alpha/(\beta C)$ to the congestion indication function [21]. However, this method is impractical, since it needs non-scalable estimation of the number of flows N inside each router. Hence, it is desirable for the router to adaptively tune $p(n)$ so that the system is free from packet loss.

One such method is AVQ (Adaptive Virtual Queue) proposed in [43, 65]. We leave the analysis of this approach under heterogeneous delays and further improvements of EMKC for future work.

4 Implementation

We next examine how to implement scalable AQM functions inside routers to provide proper feedback to MKC flows. This is a non-trivial design issue since the ideal packet loss in (85) relies on the sum of *instantaneous* rates $x_i(n)$, which are never known to the router. In such cases, a common approach is to approximate model (85) with some time-average function computed inside the router. However, as mentioned in the introduction, this does not directly lead to an oscillation-free framework since directional delays of real networks introduce various inconsistencies in the feedback loop and mislead the router to produce incorrect estimates of $X(n) = \sum_i x_i(n)$.

In what follows in this section, we provide a detailed description of various AQM implementation issues and simulate EMKC in `ns2` under heterogeneous (including time-varying) feedback delays.

4.1 Packet Header

As shown in Figure 18, the MKC packet header consists of two parts – a 16-byte *router* header and a 4-byte *user* header. The router header encapsulates information that is necessary for the router to generate precise AQM feedback and subsequently for the end-user to adjust its sending rate. The *id* field is a unique label that identifies the router that generated the feedback (e.g., its IP address). This field is used by the flows to detect changes in bottlenecks, in which case they wait for an extra RTT before responding to congestion signals of the new router. The *seq* field is a local

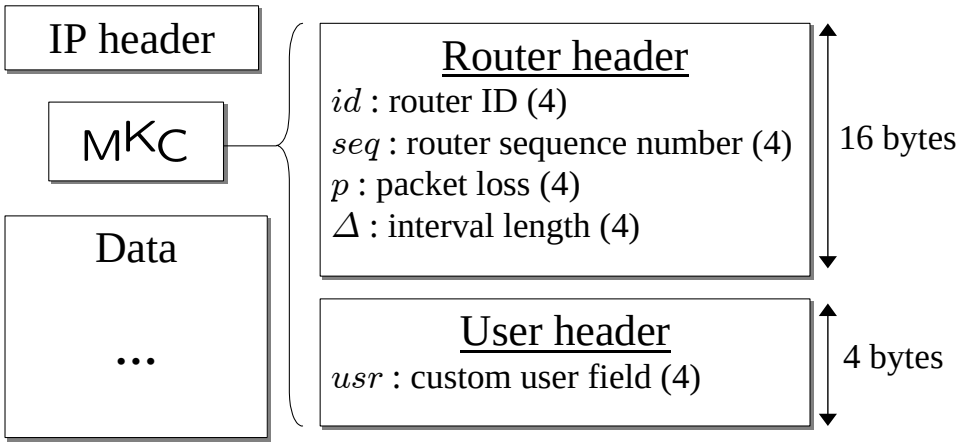


Fig. 18. Packet format of MKC.

variable incremented by the router each time it produces a new value of packet loss p (see below for more). Finally, the Δ field carries the length of the averaging interval used by the router in its computation of feedback.

The *usr* field is necessary for end-flows to determine the rate $x_i(n - D_i)$ that was in effect RTT time units earlier. The simplest way to implement this functionality is to inject the value of $x_i(n)$ into each outgoing packet and then ask the receiver to return this field in its acknowledgments. A slightly more sophisticated usage of this field is discussed later in this section.

4.2 The Router

Recall that MKC decouples the operations of users and routers, allowing for a scalable decentralized implementation. The major task of the router is to generate its AQM feedback and insert it in the headers of all passing packets. However, notice that the router never knows the exact combined rate of incoming flows. Thus, to approximate the ideal computation of packet loss, the router conducts its calculation of $p(n)$ on a discrete time scale of Δ time units. For each packet arriving within the *current*

interval Δ, the router inserts in the packet header the feedback information computed during the *previous* interval Δ. As a consequence, the feedback is retarded by Δ time units inside the router in addition to any backward directional delays $D_i^{\leftarrow}$. Since MKC is robust to feedback delay, this extra Δ time units does not affect stability of the system. We provide more implementation details below.

During interval Δ, the router keeps a local variable S, which tracks the total amount of data that has arrived into the queue (counting any dropped packets as well) since the beginning of the interval. Specifically, for each incoming packet k from flow i, the router increments S by the size of the packet: $S = S + s_i(k)$. In addition, the router examines whether its locally recorded estimate $\tilde{p}$ of packet loss (which was calculated in the previous interval Δ) is larger than the one carried in the packet. If so, the router overrides the corresponding entries in the packet and places its own router ID, packet loss, and sequence number into the header. In this manner, after traversing the whole path, each packet records information from the most congested link.[3]

At the end of interval Δ, the router approximates the combined arriving rate $X(n) = \sum_{i=1}^{N} x_i(n - D_i^{\rightarrow})$ by averaging S over time Δ:

$$\tilde{X} = \frac{S}{\Delta}. \tag{121}$$

Based on this information, the router computes an estimate of packet loss $p(n)$ using

$$\tilde{p} = (\tilde{X} - C)/\tilde{X}, \tag{122}$$

where C is the capacity of the outgoing link known to the router (these functions are

[3]Note that multi-path routing is clearly a problem for this algorithm; however, *all* existing AQM congestion control methods fail when packets are routed in parallel over several paths.

performed on a per-queue basis).

After computing $\tilde{p}$, the router increments its packet-loss sequence number (i.e., $seq = seq + 1$) and resets variable S to zero. Newly computed values seq and $\tilde{p}$ are then inserted into qualified packets arriving during the next interval Δ and are subsequently fed back by the receiver to the sender. The latter adjusts its sending rate as we discuss in the next section.

4.3 The User

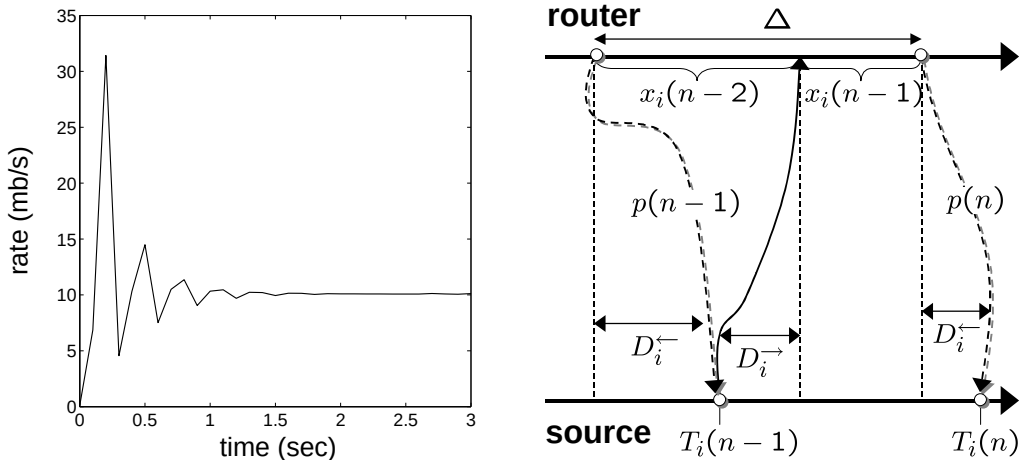
MKC employs the primal algorithm (61)-(62) at the end-users who adjust their sending rates based on the packet loss generated by the most congested resources of their paths. However, to properly implement MKC, we need to address the following issues.

First, most existing congestion control algorithms are window-based, while MKC is a rate-based method. This means that, instead of sending out a window of packets at once, each MKC user i needs to properly pace its out-going packets and maintain its sending rate at a target value $x_i(n)$. We implement this mechanism by explicitly calculating the inter-packet interval $\delta_i(k)$ of each packet k:

$$\delta_i(k) = \frac{s_i(k)}{x_i(n)}, \tag{123}$$

where $s_i(k)$ is the size of packet k of user i.

Second, notice that ACKs carrying feedback information continuously arrive at the end-user and for the most part contain duplicate feedback (assuming Δ is sufficiently large). To prevent the user from responding to redundant or sometimes obsolete feedback caused by reordering, each packet carries a sequence number seq, which is modified by the bottleneck router and is echoed by the receiver to the sender. At the same time, each end-user i maintains a local variable seq_i, which records the largest value of seq observed by the user so far. Thus, for each incoming ACK with se-



(a) (b)

Fig. 19. Naive EMKC implementation: (a) one `ns2` flow (α = 100 kb/s, β = 0.9, and Δ = 50 ms) passes through a bottleneck link of capacity 10 mb/s; (b) inconsistent feedback and reference rate.

quence seq, the user responds to it only when $seq > seq_i$. This allows MKC senders to pace their control actions such that their rate adjustments and the router's feedback occur on the same timescale.

Third, recall from (61)-(62) that MKC requires both the delayed feedback $\eta_i(n)$ and the delayed reference rate $x_i(n - D_i)$ when deciding the next sending rate. Thus, the next problem to address is how to correctly implement the control equation (61). We develop two strategies for this problem below.

4.3.1 Naive Implementation

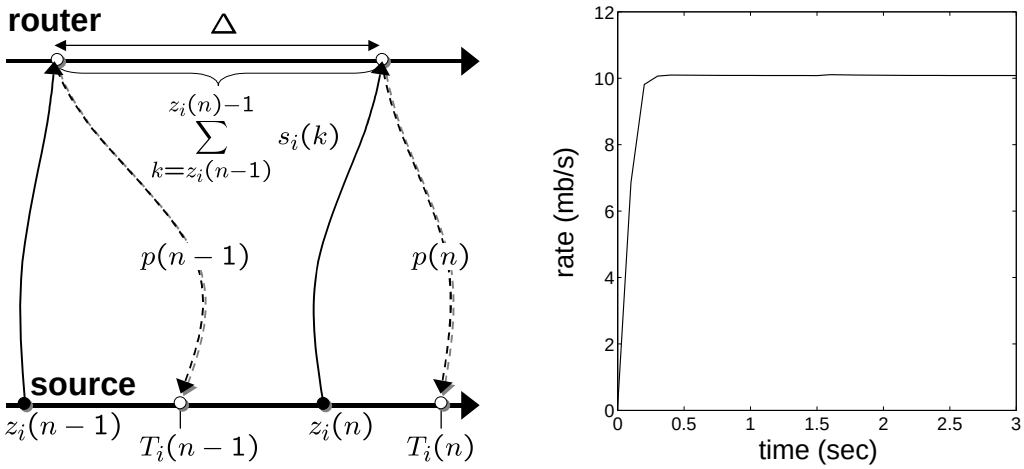
One straightforward option is to directly follow (61) based on the rate that was in effect exactly D_i time units earlier. Since round-trip delays fluctuate, the most reliable way to determine $x_i(n - D_i)$ is to carry this information in the usr field of each packet (see Figure 18). When the receiver echoes the router fields to the sender, it also copies the usr field into the acknowledgment. We show the performance of

this strategy via `ns2` simulations in Figure 19(a), in which a single MKC flow passes through a bottleneck link of capacity 10 mb/s. We set α to 100 kb/s, β to 0.9, packet size to 200 bytes, and router sampling interval Δ to 50 ms. As seen from Figure 19(a), the sending rate converges to its stationary point in less than 2 seconds and does not exhibit oscillations in the steady state; however, the flow exhibits transient oscillations and overshoots C by over 200% in the first quarter of a second. Although this transient behavior does not affect stability of the system, it is greatly undesirable from the practical standpoint.

4.3.2 Proper Implementation

To remove the transient oscillations, we first need to understand how they are created. Notice from (121) that since the router calculates the packet loss based on the average incoming rate over interval Δ, it is possible that packets of *different* sending rates $x_i(n_1)$ and $x_i(n_2)$ arrive to the router during the same interval Δ. Denote by $T_i(n)$ the time when user i receives the n-th non-duplicate feedback $p(n)$. Since the user responds to each feedback only once, it computes new sending rates $x_i(n)$ at time instances $T_i(n)$. To better understand the dynamics of a typical AQM control loop, consider the illustration in Figure 19(b). In the figure, the router generates feedback $p(n-1)$ and $p(n)$ exactly Δ units apart. This feedback is randomly delayed by $D_i^{\leftarrow}$ time units and arrives to the user at instances $T_i(n-1)$ and $T_i(n)$, respectively. In response to the first feedback, the user changes its rate from $x_i(n-2)$ to $x_i(n-1)$; however, the router observes the second rate only at time $T_i(n-1) + D_i^{\rightarrow}$. At the end of the n-th interval Δ, the router averages both rates $x_i(n-2)$ and $x_i(n-1)$ to produce its feedback $p(n)$ as shown in the figure.

When the control loop is completed, the user is misled to believe that feedback $p(n)$ refers to a single rate $x_i(n-1)$ and forced to incorrectly compute $x_i(n)$. This



(a) (b)

Fig. 20. Proper EMKC implementation: (a) graphical explanation of the algorithm; (b) one `ns2` flow ($\alpha = 100$ kb/s, $\beta = 0.9$, and $\Delta = 50$ ms) passes through a link of capacity 10 mb/s.

inconsistency is especially pronounced in the first few control steps during which flows increase their rates exponentially and the amount of error between the actual rate and the reference rate is large.

Instead of changing the router, we modify the end-users to become more sophisticated in their processing of network feedback. The key is to allow end-users to accurately estimate their own contribution to $\tilde{X}$ in (121) and determine their *average* rates seen by the router during interval Δ. For each outgoing packet k, MKC sender i places the packet's sequence number k in the *usr* field and records in local memory the size of the packet $s_i(k)$ and its sequence number k. Upon arrival of the n-th non-duplicate feedback at time $T_i(n)$, the end-flow extracts the *usr* field from the acknowledgment and records its value in variable $z_i(n)$, which is the sequence number of the packet that generated feedback $p(n)$. To compute the new rate $x_i(n)$, the user calculates the amount of data that it has transmitted between packets $z_i(n-1)$ and $z_i(n)-1$ and normalizes the sum by Δ, which is exactly the average rate used by the

router in generation of $p(n)$.

To visualize this description, consider Figure 20(a), in which the end-flow is about to decide its sending rate $x_i(n)$ at time $T_i(n)$. Notice in the figure that feedback $p(n)$ is based on all packets of flow i with sequence numbers between $z_i(n-1)$ and $z_i(n)-1$. Through the use of $z_i(n)$, we obtain a projection of the time-interval used by the router in its computation of $p(n)$ onto the sequence-number axis of the end user.[4] Given the above discussion, the user computes its average rate as:

$$\bar{x}_i(n) = \frac{1}{\Delta} \sum_{k=z_i(n-1)}^{z_i(n)-1} s_i(k), \tag{124}$$

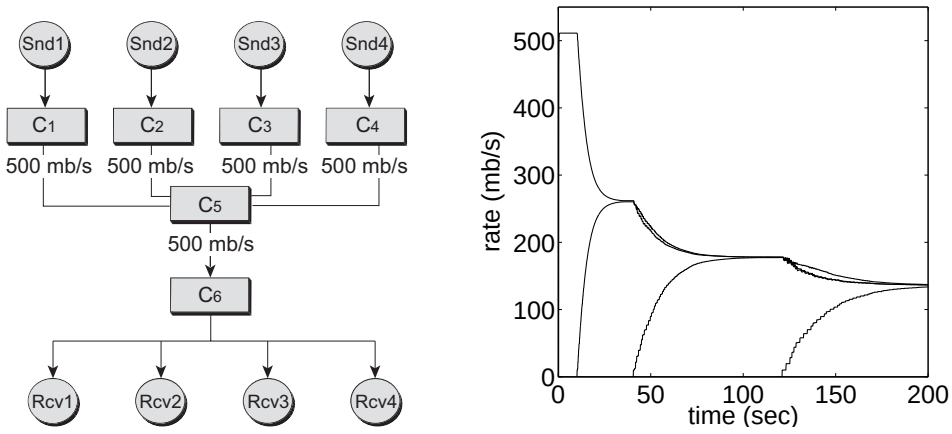
and utilizes it in its control equation:

$$x_i(n) = \bar{x}_i(n) + \alpha - \beta \eta_i(n) \bar{x}_i(n). \tag{125}$$

Next, we turn our attention to the `ns2` simulation in Figure 20(b) and examine the performance of this strategy with a single flow. The figure shows that (124)-(125) successfully eliminates transient oscillations and offers fast, monotonic convergence to the steady state.

Our next example shows the performance of the new implementation (124)-(125) with multiple flows. The simulation topology of this example is illustrated in Figure 21(a): four EMKC flows access a common bottleneck link of capacity 500 mb/s. The round-trip propagation delays of the four flows are, respectively, 10, 100, 500, and 1000 ms. As Figure 21(b) shows, flow x_1 starts with an initial rate 100 kb/s and reaches link utilization in less than 1 second. When flow x_2 joins at time 10 seconds, flow rate of x_1 is driven down toward the new stationary rate 261.1 mb/s and 99%

[4]Note that this approach is robust to random delays, but may be impeded by severe packet loss at the router.



(a) (b)

Fig. 21. Four EMKC ($\alpha = 10$ mb/s, $\beta = 0.9$, and $\Delta = 100$ ms) flows in the "dumb bell" topology.

fairness is achieved in 25 seconds. This behavior repeats as flows x_3 and x_4 start respectively at time 40 and 120 seconds, and the system quickly re-stabilizes in the new equilibrium without any transient oscillations.

4.4 Multi-Link Simulations

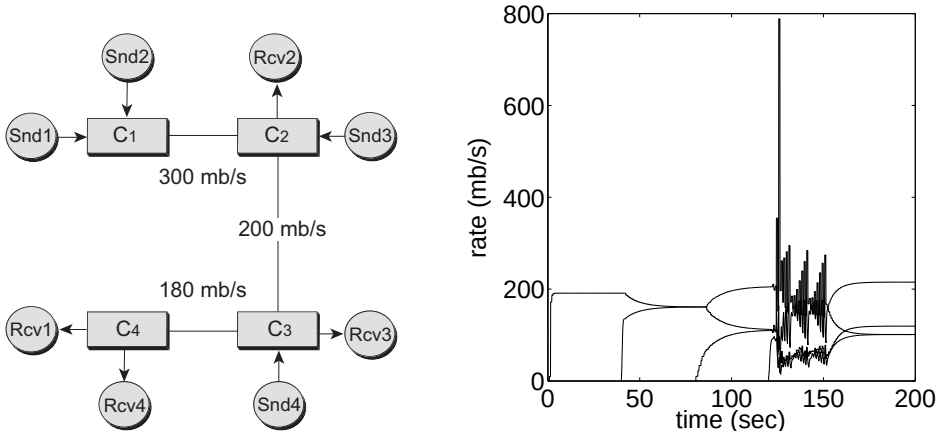
After demonstrating EMKC's single-link performance, we proceed to examine the multi-link topology illustrated in Figure 22(a). In this topology, capacities of links C_1–C_2, C_2–C_3, and C_3–C_4 are respectively 300, 200, and 180 mb/s, the corresponding round-trip propagation delays are 10, 500, and 100 ms, and the sampling intervals Δ are 100, 100, and 99 ms. In the network, there are three short flows x_2–x_4 respectively utilizing links C_1–C_2, C_2–C_3, and C_3–C_4 and one long flow x_1 passing through all three links.

The simulation result of EMKC employing implementation (124)-(125) is plotted in Figure 22(b). Flow x_1 starts first and reaches the utilization of the bottleneck link C_3–C_4 in 2 seconds. As flow x_2 joins at time 40 seconds, the bottleneck of x_1 switches

to link C_1–C_2 and both x_1 and x_2 converge to fairness. Similarly, when x_3 starts at time 80 seconds, link C_2–C_3 becomes the new bottleneck of x_1. As a consequence, x_1 and x_3 converge toward the new fair rate and x_2 climbs up and collects the residual bandwidth on link C_1–C_2. The individual sending rates are smooth during the first 120 seconds. However, after flow x_4 joins, the system suffers sustained oscillations during the period from time 124 through 151 seconds. The oscillation is especially severe for x_1, whose sending rate reaches as high as 788.5 mb/s at time 125.7 seconds, overshooting the bottleneck C_3–C_4 by over 300%.

We next investigate the underlying reason for this oscillation. Observe that as the sending rate of x_4 increases, the bottleneck of flow x_1 switches from link C_2–C_3 to C_3–C_4. Since it is possible for this switching to occur in the middle of the bottleneck router's sampling interval, the computed packet loss could be inconsistent with the end-user's reference rate. This results in fluctuations in the sending rate and is the primary reason for the "spike" shown in Figure 22(b). Moreover, this situation could be exacerbated when multiple resources with close capacities (e.g., 180 and 200 mb/s in this case) exist in the path of a certain user, since fluctuating input rate at the routers will cause fluctuating packet loss, which could eventually lead to oscillations of the bottleneck link and aggravate the rate oscillations of the end-users. This explains the oscillations after the spike.

We emphasize that these problems do not indicate instability of EMKC, but arise as the result of discretized implementation of the theoretical model given by (63) and (85). To properly deal with multi-bottleneck networks, we develop several strategies to manage bottleneck switching. First, we force the end-user to delay its response to the ACK for one RTT once a bottleneck switch is detected. By doing this, the packet loss carried in the next non-duplicate ACK will be consistent with the reference rate $\bar{x}_i(n)$ computed by the user. Second, we damp the bottleneck oscillations resulting



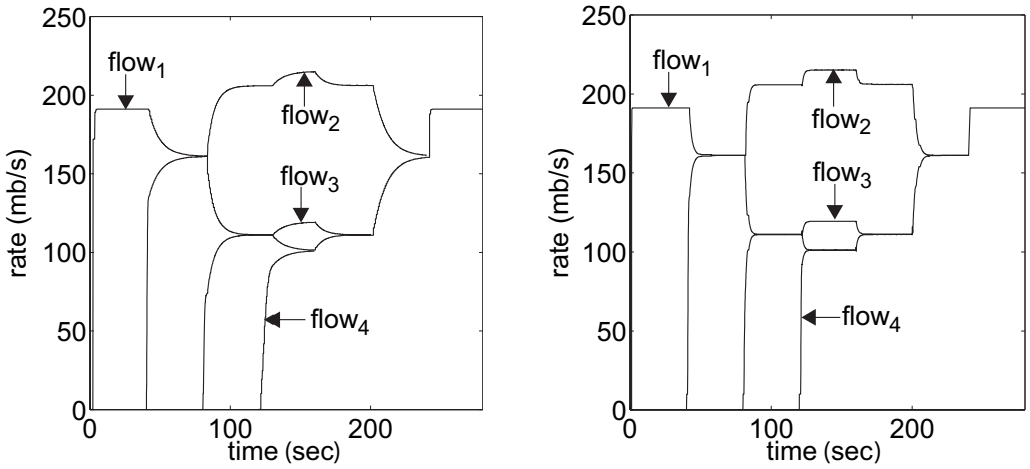
(a) (b)

Fig. 22. (a) "Parking lot" topology; (b) Naive implementation of EMKC ($\alpha = 10$ mb/s and $\beta = 0.9$) in the "parking lot" topology.

from multiple routers with close capacities by introducing a threshold value δ such that the end-user authorizes a bottleneck switch if only if the difference between the old packet loss and the packet loss carried in the ACK is greater than δ.

To examine the effectiveness of this mechanism, we redo the simulation in Figure 22(b) using this new algorithm and plot the result in Figure 23(a). As seen in the figure, this implementation removes the oscillations that originally occurred when x_4 joined the system. Starting from time 160 seconds, flows x_4, x_3, and x_2 terminate with a 40-second delay, and there is no oscillation in both the transient phase and the steady state.

We next incorporate randomness into the feedback delay of individual flows and test EMKC in settings with highly variable delays. To implement time-varying delay, we maintain a local queue at the receiving end of each flow and force the ACKs to pass through this queue before being echoed to the sender. For every m successfully transmitted acknowledgments, the system delays the head packet in the queue by d seconds and the other packets by 10 μs. Here, time-varying variables d and m



(a) constant delay (b) random delay

Fig. 23. Proper implementation of EMKC ($\alpha = 10$ mb/s, $\beta = 0.9$, and $\delta = 0.01$) in the "parking lot" topology.

are uniformly distributed in $[0.5, 1]$ and $[500000, 1000000]$, respectively. All packets between the d-second delay spikes are drained at the wire speed of the return path, which ensures that the queue is completely emptied before the next spike is generated. We preserve the topology in Figure 22(a) except that the round-trip propagation delay of each flow is fixed to be 10 ms such that the effect of random delay is more evident. The simulation result is depicted in Figure 23(b), in which the system exhibits delay-independent asymptotic stability, fast convergence to the stationary point, and smooth transitions between the neighboring states.

5 Discussion

This chapter investigated the properties of Internet congestion controls under non-negligible directional feedback delays. We focused on the class of control methods with radial Jacobians and showed that all such systems are stable under heterogeneous delays. To construct a practical congestion control system with a radial (symmetric

in particular) Jacobian, we made three changes to the classic discrete Kelly control and created a max-min version we call MKC. Combining the latter with a negative packet-loss feedback, we developed a new controller EMKC and showed in theory and simulations that it offers smooth sending rate and fast convergence to efficiency. Furthermore, we demonstrated that EMKC's convergence rate to fairness is exponential when the network provider scales the number of flows N as $\Theta(C)$ and linear otherwise. From the implementation standpoint, EMKC places very little burden on routers, requires only two local variables per queue and one addition per arriving packet, and allows for an easy implementation both in end-to-end environments and under AQM support. However, EMKC has two key limitations – constant packet loss in the steady state and slow (linear) convergence rate to fair resource allocation (i.e., fairness). We next overcome both drawbacks by designing a new protocol called JetMax.

CHAPTER VI

JETMAX

1 Introduction

In the light of TCP's scalability issues in high-speed networks [30], explicit-feedback congestion control has gained renewed interest in the last several years [56, 74, 114, 115]. Sometimes referred to as *Active Queue Management (AQM) congestion control*, these algorithms rely on routers to provide congestion feedback in the form of changes to the congestion window [56], packet loss [115], single-bit congestion indication [44, 62, 88], queuing delay [49, 107], or link prices [60, 66, 77]. This information helps end-flows converge their sending rates to some social optimum and achieve a certain optimization objective.

Unlike some of the largely ineffective AQM aimed at improving the performance of TCP [20], properly designed explicit congestion control promises to provide scalability to arbitrary bandwidth (i.e., terabits and petabits per second[1]), tunable link utilization, low delay, zero loss, oscillation-free steady state, and exponential convergence to fairness/efficiency, all of which suggests that such algorithms, once deployed in the Internet, may remain in service for many years to come. Note that the purpose of this work is not to settle the debate of whether or when explicit congestion control will be adopted by the Internet, but to explore the various properties of existing AQM methods, propose a new controller we call JetMax, and compare its `ns2` and Linux performance with that of the existing methods.

[1] If network bandwidth continues to double every year, these speeds will become mainstream in 10 and 20 years, respectively.

The first half of this chapter deals with understanding delayed stability and convergence performance of several recently proposed AQM approaches: eXplicit Control Protocol (XCP) [56], Rate Control Protocol (RCP) [26], Exponential Max-min Kelly Control (EMKC) [115], and a hybrid method suggested in [115] that combines EMKC with Adaptive Virtual Queue (AVQ) [66, 65]. We find from this study that both XCP and RCP are sensitive to RTT estimation and is prone to instability even in single-link topologies where the average RTT d_l estimated by the router is significantly different from the maximum RTT D_{max} of end-flows. Although this issue can be overcome by utilizing D_{max} instead of d_l, additional problems may occur under time-varying delays. Moreover, XCP may become unstable in certain multi-link networks when the flows receive feedback on different time scales (i.e., under heterogeneous delay). The root of this problem lies in the oscillatory switching between the bottlenecks (i.e., changes in the bottleneck link) and inability of each XCP flow to permanently decide its most-congested resource in the presence of delayed feedback. This phenomenon in turn arises from the *discontinuous* nature and *non-monotonic* transient properties of the feedback function used in the control equation of XCP. Discontinuity of feedback follows from XCP's algorithm for selecting the most-congested link along its path, while non-monotonicity is caused by the oscillatory nature of the controller when the feedback delays of competing flows are heterogeneous.

To further understand the reasons for XCP's instability in multi-link networks, we analyze the problem of bottleneck oscillation in more depth and show that only *consistent* (i.e., agreed upon by every flow) bottleneck assignment allows one to reduce stability analysis of max-min protocols in multi-link networks to that of the single-link case studied in prior work [26, 56, 107, 115]. In all other cases, max-min methods require a much more complicated analysis not available within the current framework of congestion control. We additionally observe that feedback that remains *monotonic*

when a flow changes its most-congested resource allows the protocol to achieve a consistent bottleneck assignment and thus remain stable. This partially explains EMKC's stability in multi-link networks observed in simulations.

Although EMKC remains stable in multi-link topologies, we find that its transient and equilibrium properties (such as linear convergence to fairness and steady-state packet loss) are potential drawbacks for its use in practice. The problem of EMKC's equilibrium packet loss can be overcome using EMKC-AVQ; however, the resulting method exhibits undesirable oscillations and transient overshoot of link's capacity. Combined with a large number of flows, transient overshoot leads to long-lasting packet loss and non-negligible increase in queuing delay, both of which are highly undesirable.

Our conclusion from the first half of the chapter is that any new designs of max-min AQM congestion control should decouple feedback delay from control equations and converge to stationarity monotonically. Thus, the second part of this chapter designs a new method we call JetMax that satisfies these criteria while offering additional features:

- *Capacity-independent convergence time.* The algorithm reaches fairness and efficiency in the *same* number of RTT steps regardless of link's capacity.
- *Zero packet loss.* Loss-free operation is ensured both in the transient and stationary state.
- *Tunable link utilization.* Each router can be *independently* configured to control its steady-state link utilization.
- *RTT-independent max-min fairness.* Resource allocation is max-min fair regardless of end-user delays.
- *Global multi-link stability under consistent bottleneck assignment for all types*

of delay. Flows converge to the equilibrium and maintain their steady-state rates in generic networks regardless of any fluctuation in the RTT as long as end-users can correctly choose their bottleneck links (see below for more).

- *Low overhead.* The AQM algorithm requires only three additions per arriving packet and *no* per-flow state information inside routers.

We finish the chapter by repeating the same `ns2` simulations that earlier highlighted the limitations of existing methods and demonstrate that JetMax outperforms its predecessors using a number of metrics such as multi-link stability, convergence rate, transient overshoot, and steady-state rate allocation. We also show that JetMax can be easily integrated into the Linux router kernel and present the results of Linux experiments with JetMax running over 1 gb/s links, both in single- and multi-bottleneck topologies.

2 Background

We start by describing the notation used throughout the chapter. Assume N users in the network whose rates at time t are given by $\{x_r(t)\}_{r=1}^N$. Following notation in [51], we denote the RTT of each flow by $D_r(t)$ and the forward/backward delays of user r to/from link l by $D_{r,l}^{\rightarrow}(t)$ and $D_{r,l}^{\leftarrow}(t)$, respectively. The aggregate arrival rate of all users at link l is written as $y_l(t) = \sum_{r \in l} x_r(t)$, where $r \in l$ is the set of flows r passing through link l. Similarly, notation $l \in r$ refers to the set of links l used by flow r.

Since its appearance in 2002, XCP [56] has become a de-facto standard for explicit congestion control in IP networks [29]. XCP is a window-based framework, in which routers continuously estimate aggregate flow characteristics (e.g., arrival rate, average RTT) and feed back the desired changes to the congestion window to each

bottlenecked flow through its packet headers. Stability of XCP under heterogeneous delay is unknown at this time; however, for homogeneous delay $D_r(t) = D$, the paper shows that the combined rate $y_l(t)$ is stable if $0 < \alpha < \pi/4\sqrt{2}$ and $\beta = \alpha^2\sqrt{2}$, where α and β are constants used in the XCP control equation.

XCP's design goals [56] include max-min fairness and high link utilization; however, a recent study of its equilibrium properties [74] shows that XCP does not generally achieve max-min fairness in multi-link networks and its link utilization may sometimes be as low as 80%. The paper further demonstrates scenarios where XCP allocates arbitrarily small (unfair) fractions of bandwidth to certain flows [74]. Another study [114] reports experiments with a 10-mb/s XCP Linux router and identifies several implementation issues including uncertainty in accurate selection of link's capacity, sensitivity to receiver buffer size, and various problems with partial deployment.

The recently proposed Rate Control Protocol (RCP) [26] is a rate-based max-min AQM algorithm in which each link l periodically computes the desired sending rate $r_l(t)$ for flows bottlenecked at l and inserts $r_l(t)$ into their packet headers. This rate is overridden by other links if their suggested rate is less than the one currently present in the header. Links decide the fair rate $r_l(t)$ by implementing a controller

$$r_l(t) = r_l(t - \Delta)\Big[1 - \frac{\Delta}{d_l C_l}\Big(\alpha(y_l(t) - C_l) - \beta\frac{q_l(t)}{d_l}\Big)\Big], \tag{126}$$

where Δ is the router's control interval, α and β are constants, d_l is a moving average of RTTs sampled by link l, C_l is its capacity, and $q_l(t)$ is its queue length at time t. Compared to XCP, RCP has lower implementation overhead, offers quicker transient dynamics, and achieves max-min fairness [26].

Two additional max-min methods are inspired by Kelly's optimization framework [60] and aim to improve stability and convergence properties of traditional models of

additive packet loss [65, 77]. The first approach called MaxNet [107] obtains feedback $f_r(t) = \max_{l \in r} p_l(t)$ from the most congested link along each path of user r and applies an unspecified end-user control function to $f_r(t)$ so as to converge the sending rates of all flows to max-min fairness. To avoid equilibrium packet loss, link prices are driven by a controller

$$\dot{p_l}(t) = \frac{y_l(t) - \gamma C_l}{C_l}, \tag{127}$$

where $0 < \gamma < 1$ is the desired link utilization.

The second method is Exponential Max-min Kelly Control (EMKC) [115], which elicits packet-loss from the most-congested resource along each flow's path and uses a modified version of the discrete Kelly equation to achieve delay-independent stability. End-user rates $x_r(n)$ are adjusted using

$$x_r(n) = x_r(n - D_r) + \alpha - \beta p_r(n) x_r(n - D_r), \tag{128}$$

where ${D_r}^2$ is the RTT of flow r, $\alpha > 0$ and $0 < \beta < 2$ are constants, and $p_r(n) \in (-\infty, 1)$ is the packet-loss feedback received by flow r at time n. The feedback function allows negative values and assumes the following shape [115]

$$p_r(n) = \max_{l \in r} \frac{\sum_{s \in l} x_s(n - D^{\rightarrow}_{s,l} - D^{\leftarrow}_{r,l}) - C_l}{\sum_{s \in l} x_s(n - D^{\rightarrow}_{s,l} - D^{\leftarrow}_{r,l})}. \tag{129}$$

We remark that throughout the chapter, we use the same time unit (say, ms) for time n, delay D_r, and control interval Δ. Therefore, all these metrics are assumed to be integers.

For a single-link network, system (128)-(129) is locally asymptotically stable for all time-varying delays. Due to the steady-state overshoot of link's capacity [115],

[2]Since all MKC-based methods examined in the chapter and the later introduced scheme JetMax do not estimate the RTT and are delay-independent, we replace time-varying delay $D_r(t)$ by its constant version D_r for ease of presentation.

EMKC does not reach max-min fairness. However, as suggested in [115], EMKC can be combined with AVQ [66] to guarantee max-min fair rates and zero loss in the stationary state.

3 Understanding Existing Methods

This section discusses the desired properties of future congestion control and examines whether the existing methods satisfy these requirements. We focus on such issues as flow dynamics under heterogeneous (both time-invariant and time-varying) feedback delay, stability in multi-link scenarios, convergence behavior, and overshoot properties in transient and equilibrium states.

3.1 Ideal Congestion Control

During the design and analysis of congestion control, many issues are taken into consideration; however, one of the most fundamental requirements on modern congestion control is its asymptotic stability under heterogeneous (including time-varying) delays. The reason we focus on non-deterministic delay is to understand the various deployment issues that a protocol may face in real networks, where the forward delay between the source and each link, as well as the corresponding backward feedback delay, are dynamic (often random) metrics [84]. Traditional models of congestion control [56, 65, 77, 100] usually assume a certain "determinism" about the RTT (i.e., queuing delays are either fixed or based on fluid approximations) and sometimes produce results that no longer hold under more realistic conditions [71]. It thus becomes important to examine how protocols behave in highly heterogeneous environments and whether fluctuating feedback delay may cause them to oscillate.

Besides stability, ideal congestion control should exhibit fast convergence to both

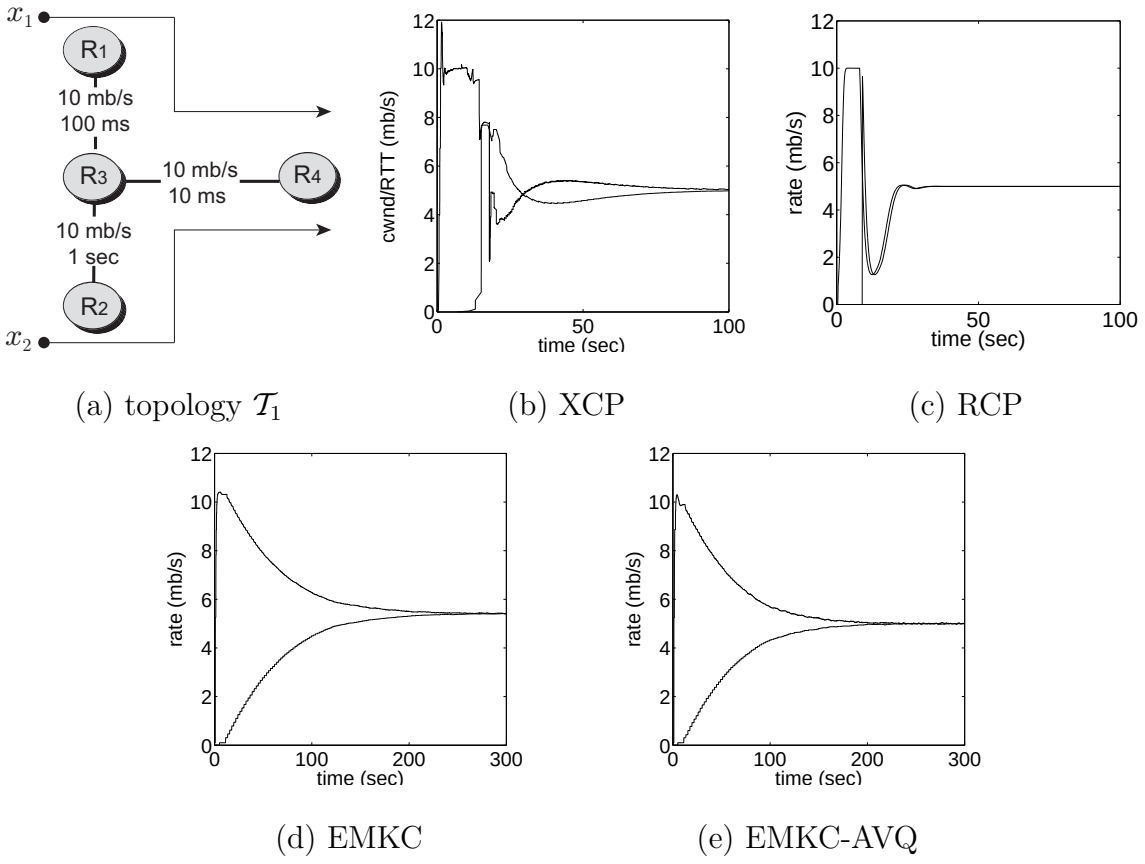
efficiency and fairness, avoid overshooting capacity in transient and stationary states, and converge to the desired link utilization γ. While the first few factors are mostly important to end-users, the last metric is of interest to network operators, who usually run their backbones at well below capacity and may not appreciate protocols (such as [49, 56]) that always try to achieve 100% utilization.

Our results below show that none of the existing methods satisfies all of these requirements simultaneously. Some protocols exhibit oscillations and instability under heterogeneous RTTs or in certain multi-link topologies, while others demonstrate undesirable stationary and/or transient properties. As a result of this study, we first come to understand the need for and then develop a new method that is capable of simultaneously meeting the design criteria above while admitting a simple implementation inside routers.

3.2 Methodology

Our main focus in this comparison study is on XCP [56], RCP [26], and EMKC [115] as completely different approaches to max-min congestion control. At the time of this writing, MaxNet [107] did not have a publicly available `ns2` implementation; however, we found that a combination of EMKC and AVQ [66] possessed transient and stationary behavior similar to that of MaxNet. Recall that AVQ dynamically adjusts the virtual capacity of each link until the arrival rate $y_l(t)$ is stabilized at γC_l, where γ is the desired link utilization. This method is similar to the price integrator (127) in MaxNet with the exception that AVQ is not feedback-specific.

Throughout this section, we use `ns2` simulations with AVQ code that comes with the simulator (version 2.27), and XCP, RCP, and EMKC code used in [56, 26, 115], respectively. We also experimented with the modified XCP code from ISI [108] and found it to offer no stability benefits over the original code. We thus limit our XCP



(a) topology $\mathcal{T}_1$ (b) XCP (c) RCP

(d) EMKC (e) EMKC-AVQ

Fig. 24. XCP, RCP, EMKC, and EMKC-AVQ in topology $\mathcal{T}_1$.

discussion to the algorithms used in [56].

We should finally emphasize that simulation scenarios shown below are meant to highlight the possibility of unstable behavior and demonstrate the undesirable convergence properties of the studied protocols rather than providing their exhaustive evaluation under "realistic" Internet conditions.

3.3 Stability under Heterogeneous Delay

We first study how each method handles heterogeneous delay over a single link. We use topology $\mathcal{T}_1$ shown in Fig. 24(a), where two flows x_1 and x_2 with round-trip delays 220 and 2020 ms, respectively, start with a 5-second delay and share a 10-mb/s link.

For XCP, we use the parameters suggested in [56] (i.e., $\alpha = 0.4$ and $\beta = 0.226$) and set the buffer size sufficiently large (i.e., at least $C_l \times RTT$). As Fig. 24(b) shows, XCP is stable under heterogeneous delay, even though it exhibits oscillations and relatively slow (compared to the case of homogeneous D) convergence to fairness.

We next examine RCP with $\alpha = 0.4$ and $\beta = 1.0$, whose simulation result[3] is given in Fig. 24(c). As seen in the figure, RCP exhibits stable behavior and converges the sending rates to the fair share of the bottleneck bandwidth. However, we can also observe from the figure the "spike" in x_2's sending rate as it joins the system. This results in instantaneous overshoot of the link capacity and buildup of queue backlog in the router. This problem becomes progressively serious in the presence of multiple arriving flows, in which case any buffer can be overflow by a sufficiently large number of new users.

For EMKC we set $\alpha = 0.2$ mb/s, $\beta = 0.5$, and control interval $\Delta = 100$ ms, and repeat the simulation in $\mathcal{T}_1$. The result is plotted in Fig. 24(d), which demonstrates that EMKC converges to the stationary state much more smoothly than XCP and RCP; however, it spends over 250 seconds before reaching fairness and eventually overshoots link's capacity by 8%. Although EMKC's convergence rate can be improved by increasing α, this leads to more steady-state packet loss and larger overshoot [115]. We delay further discussion of this issue until later in the section.

The fourth method to examine is the combination of EMKC and AVQ. We experimented with the default `ns2` code of AVQ, but found it to be too noisy due to the random fluctuations in inter-packet arrival delays and the fact that AVQ estimates $y_l(t)$ on a per-packet basis. To make the method actually *converge* to its stationary

[3]For all rate-based methods examined in the chapter (i.e., RCP, EMKC, EMKC-AVQ, and JetMax), we refer to "rate" as the sending rate. In addition, since JetMax does not build up queues or drops packets, its sending rate equals the receiving rate.

state, we modified AVQ to estimate the aggregate input rate $y_l(n)$ every Δ time units and adjust the virtual capacity $\tilde{C}_l$ at the end of this interval:

$$\tilde{C}_l(n) = \tilde{C}_l(n-\Delta) + \frac{\tau\Delta(\gamma C_l - y_l(n))}{D_{max}}, \tag{130}$$

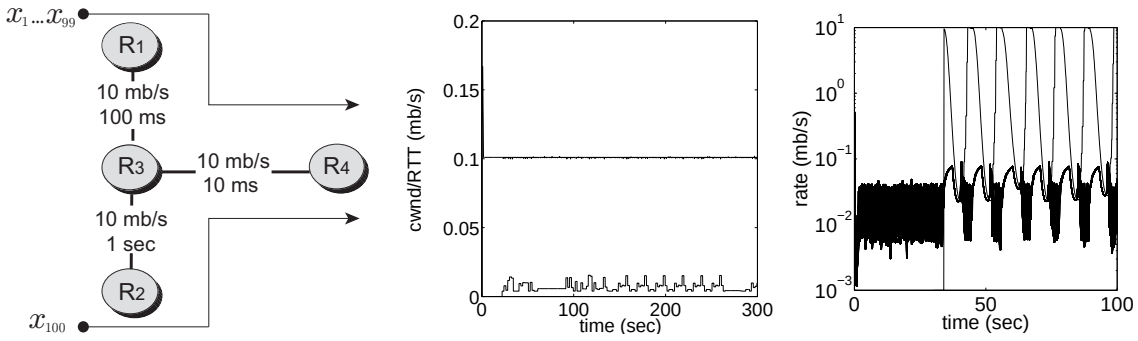
where $\tau = 0.2$ is the gain parameter used throughout this chapter, γ is the desired link utilization, D_{max} is the maximum RTT of end-flows, and C_l is the true capacity of the link.[4] It is not difficult to notice that (130) is in fact an Integral controller [113] on virtual capacity $\tilde{C}_l(n)$ and converges combined rate $y_l(n)$ to its target value γC_l. The final step of EMKC-AVQ is to limit $\tilde{C}_l$ to the range $(-\infty, \gamma C_l]$ and then apply its value in (129) to compute the feedback. Using this implementation, we repeat the above simulation and plot the result in Fig. 24(e), which indicates that EMKC-AVQ is indeed max-min fair in the steady state (i.e., both flows achieve 5 mb/s) as well as stable under heterogeneous delays; however, the convergence rate to fairness remains painfully slow (i.e., over 200 seconds).

3.4 Sensitivity to RTT Estimation

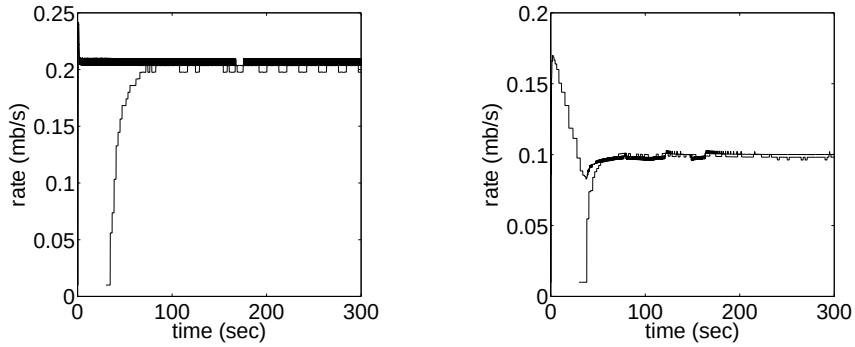
The situation can become complicated by slight modifications of topology $\mathcal{T}_1$, in which flow x_1 is replaced by a group of 99 flows x_1-x_{99}. Simulation results of these methods in this new topology $\mathcal{T}_2$ are given in Fig. 25.

As seen from Fig. 25(d) and 25(e), both EMKC and EMKC-AVQ are stable in this scenario and converge their sending rates to the expected stationary values. On the contrary, neither XCP nor RCP is stable and XCP even exhibits a denial-of-service effect on flow x_{100}. Instability of these two protocols arises when the *average*

[4] All delays are computed using XCP's smoothed EWMA estimator with the default weight 0.4 and (130) is normalized by D_{max} to ensure stability of the resulting system under delayed feedback [65].



(a) topology $\mathcal{T}_2$ (b) XCP (c) RCP



(d) EMKC (e) EMKC-AVQ

Fig. 25. XCP, RCP, EMKC, and EMKC-AVQ in topology $\mathcal{T}_2$.

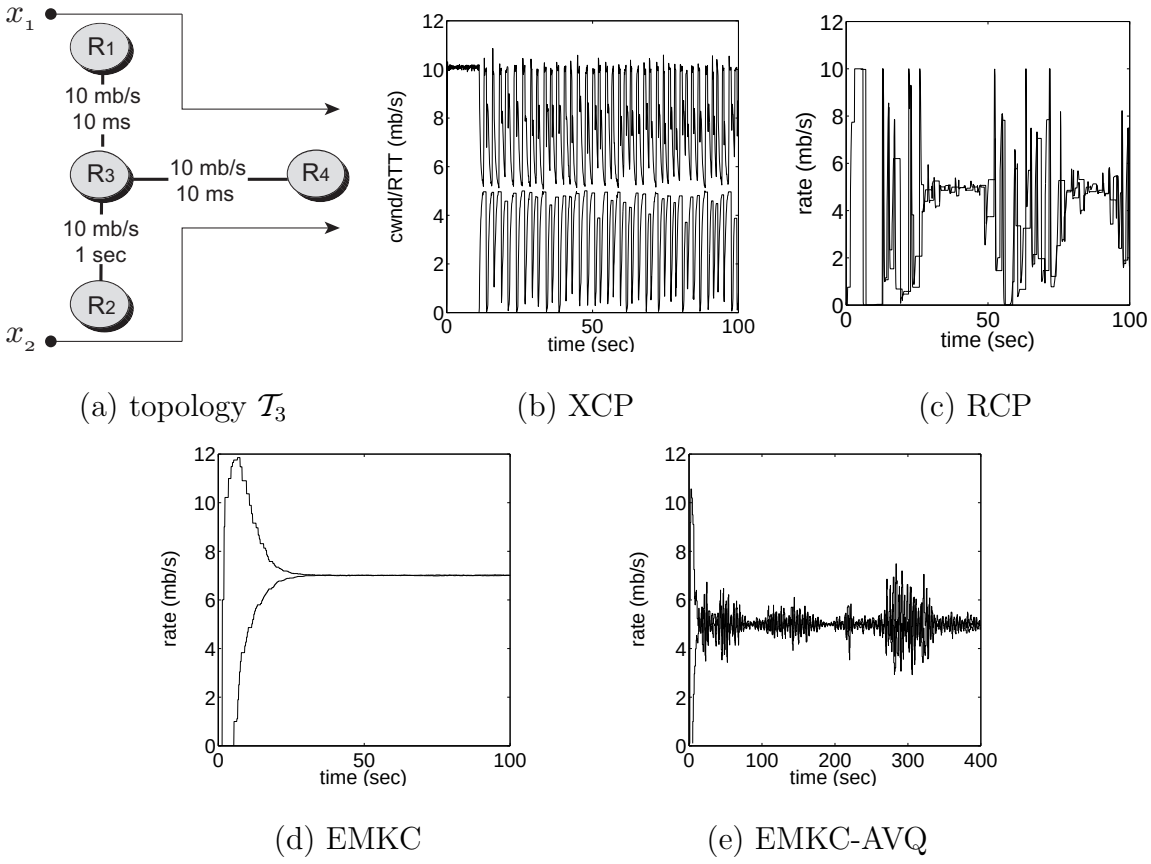
RTT d_l significantly deviates from the *maximum* RTT D_{max} of all flows. Notice that in topology $\mathcal{T}_2$, the average delay measured by the bottleneck router is only 200 ms, while the "slowest" flow x_{100} receives feedback with a 2-second delay. This results in unstable oscillations shown in Fig. 25(b) and 25(c).

An obvious solution to this problem is to utilize D_{max} instead of d_l in XCP and RCP's control equations. We changed `ns2` packet headers to carry the smoothed RTT of each end-flow and adapted XCP and RCP's router code to use the maximum RTT observed in any control interval instead of d_l . The resulting systems did in fact exhibit expected performance and were stable in $\mathcal{T}_2$ (results not shown for brevity). Nevertheless, this change does not solve all XCP and RCP's problems related to delay.

3.5 Time-Varying Delay

Although using the maximum RTT D_{max} is effective under *fixed* heterogeneous delays, it may have problems when the RTT is time-varying. This can be demonstrated with the help of topology $\mathcal{T}_3$ illustrated in Fig. 26(a), where we generate random feedback delays by forcing the receiver to pass its acknowledgments through a local queue, which randomly delays the packets before sending them to the source. The algorithm applies a random d-second delay-spike to the head packet of the queue every m successfully transmitted acknowledgments and delays the remaining $m-1$ packets by 10 μs, where d and m are uniformly distributed in $[0.5, 1.0]$ and $[5000, 10000]$, respectively. This delay pattern ensures that the queue is completely emptied before the next spike and approximates periodic congestion in the Internet caused by flash crowds, routing changes, and oscillatory behavior of cross-traffic flows.

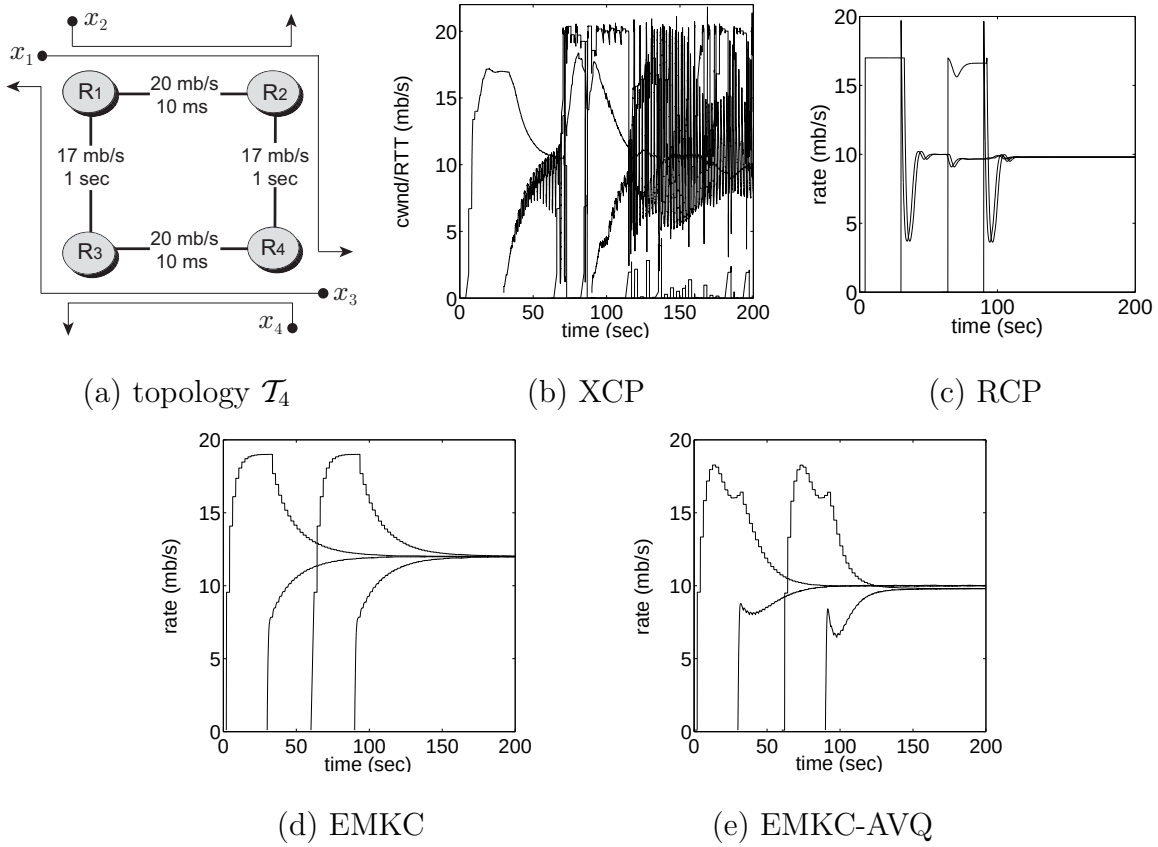
From Fig. 26, we can see that EMKC is the only stable method in this variable-delay scenario since it does not rely on RTT estimation and its stability is delay-independent. Instability of XCP, RCP and EMKC-AVQ (all of which use D_{max} in the



(a) topology $\mathcal{T}_3$ (b) XCP (c) RCP

(d) EMKC (e) EMKC-AVQ

Fig. 26. XCP, RCP, EMKC, and EMKC-AVQ in topology $\mathcal{T}_3$.



(a) topology $\mathcal{T}_4$ (b) XCP (c) RCP

(d) EMKC (e) EMKC-AVQ

Fig. 27. XCP, RCP, EMKC, and EMKC-AVQ in multi-link topology $\mathcal{T}_4$.

router equations) arises from delay fluctuation, since by the time the router "learns" the new D_{max}, it may become out-dated in the next control interval and the system is already unstable. Additional filters and fixes may make these methods stable in this scenario; however, our next set of simulations show that a more fundamental problem prevents XCP (and potentially other max-min methods) from operating well in highly heterogeneous networks.

3.6 Multi-link Stability

Our next stability issue is to examine the performance of these protocols in multi-link networks where bottlenecks shift over time and there exists a possibility for incorrect

inference of the most-congested link. For the purpose of this section, we study the four-bottleneck case $\mathcal{T}_4$ shown in Fig. 27(a), where four flows $x_1, \ldots, x_4$ are routed over a grid-type network. We customize the routing rules at nodes R_1 and R_4 to always route their traffic (including any ACKs) in the clockwise direction. This ensures that acknowledgments of flow x_1 travel together with flow x_3 and vice versa. At the same time, the acknowledgments of flows x_2 and x_4 are routed along their corresponding shortest paths (i.e., $R_2 - R_1$ and $R_3 - R_4$). Flows start in sequence from x_1 to x_4 with a 30-second delay. Given this order of user join, flow x_1 should originally converge to 17 mb/s and shifts its bottleneck to accommodate flow x_2. The same expected behavior also applies to flows x_3 and x_4. The final max-min assignment of rates is 10 mb/s for each flow.

Fig. 27(b) shows the behavior of XCP in $\mathcal{T}_4$. Notice in the figure that the protocol not only oscillates for over 200 seconds, but also denies service to flow x_3, which never obtains its share of the link even in the average sense. The reason for oscillation can be traced to the fact that both x_1 and x_3 continuously switch between their bottlenecks and are unable to settle down in the selection of their most-congested link. This is caused by non-monotonicity of feedback at each link, discontinuous control actions of end-users, and random fluctuation of the RTT that forces XCP to become unstable on small timescales. In contrast, RCP in Fig. 27(c), EMKC in Fig. 27(d), and EMKC-AVQ in Fig. 27(e) have no visible stability problems and converge their sending rates exactly as expected.

3.7 Convergence Speed

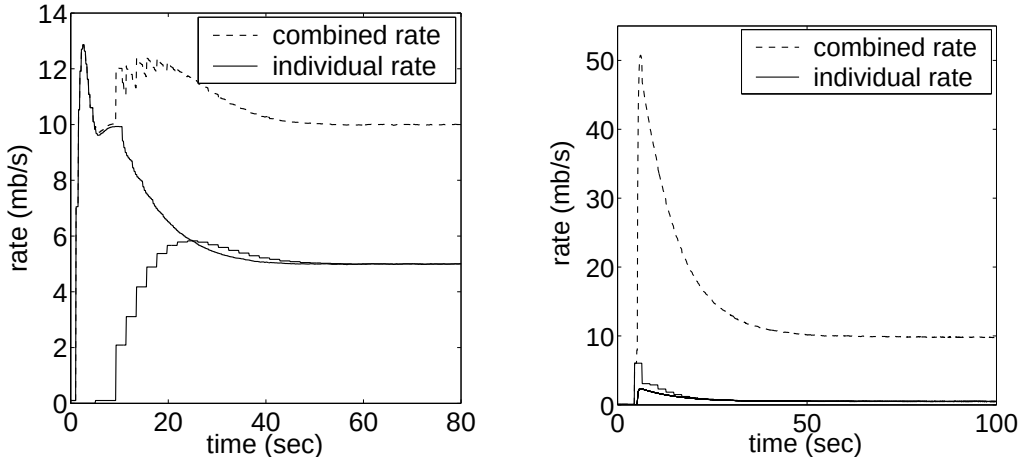
Besides stability, another metric we evaluate is the convergence speed to stationarity. XCP generally converges quickly over links with homogeneous delay; however, its convergence rate may be compromised by heterogeneity of delay and oscillations of

the controller inside routers. One example of this behavior is shown in Fig. 24(b), where it takes XCP over 1.5 minutes to reach fairness on a 10 mb/s link. At the time of this writing, there are no known expressions for XCP's convergence rate to efficiency or fairness and future analysis of these metrics appears difficult due to the complex behavior of the controller under delay.

As to the best of our knowledge, there does not exit an explicit expression of RCP's convergence speed. However, we can empirically observe that RCP, in its stable cases, exhibits the best convergence properties among all methods studied in this section. In both Fig. 24(c) and 27(c), it takes RCP around 30 seconds (i.e., 15 RTTs) to reach the stationarity.

For EMKC and small $N\alpha \ll C$, [115] shows that flows reach fairness in $\Theta(C \log N/(N\alpha))$ steps, which scales *linearly* with resource capacity C. In Fig. 24(c), for instance, it takes two EMKC flows over 4 minutes to reach fairness on a 10-mb/s link. Furthermore, the major problem with EMKC's convergence rate to fairness is the tradeoff between convergence speed and stationary packet loss in the network. For small fixed α, EMKC's linear rate of convergence is clearly undesirable, especially in high-speed networks. To achieve capacity-independent convergence, α must be on the order of C, which results in large stationary packet loss since the amount of steady-state overshoot $N\alpha/\beta$ is now comparable to C [115]. In general, there is no algorithmic way for end-flows to select their α so as to keep loss low and convergence to fairness quick. This is one of the main drawbacks of EMKC.

Similar arguments apply to EMKC-AVQ. Even though it does not suffer from steady-state packet loss, as we show next, EMKC-AVQ's transient packet loss that is proportional to α keeps the protocol from quickly converging to fairness.



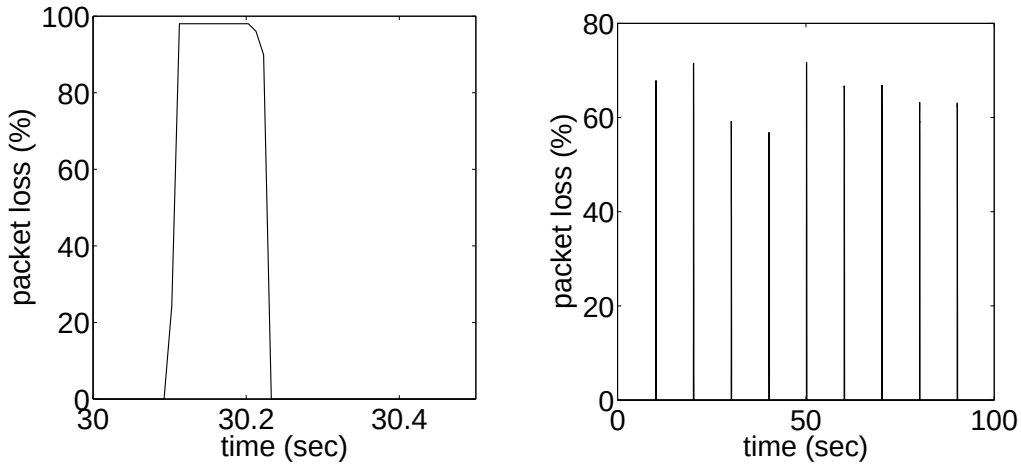
(a) $N = 2$ (b) $N = 20$

Fig. 28. Transient overshoot of EMKC-AVQ ($\alpha = 2$ mb/s, $\beta = 0.5$ and $\tau = 0.2$).

3.8 Overshoot Properties

Another issue to consider when designing congestion control is the amount of overshoot and oscillation before the stationary state is reached. For discussion purposes below, we semantically equate overshoot of network capacity with packet loss, even though small overshoots (in terms of amount and/or duration) can often be absorbed by buffers and do not necessarily lead to packet loss. Nevertheless, we aim to stress that any overshoot (especially by 10000 concurrent flows) leads to stressful conditions at the router and, in the least, increases the queuing delay. In addition, depending on how long the feedback is delayed on the way to the sender, any "innocent" overshoot of C may lead to substantial packet loss and create a hostile environment for other flows.

Among the four controllers in this comparison study, XCP, according to the simulations, does not encounter a severe challenge imposed by transient overshoot. In contrast, EMKC has the worst equilibrium properties since its combined stationary rate $y^* = C + N\alpha/\beta$ is strictly above the bottleneck capacity C. Moreover, this packet



(a) (b)

Fig. 29. Transient overshoot of RCP ($\alpha = 0.4$ and $\beta = 1$).

loss scales linearly with the number of connections and becomes worse if one increases α to accelerate the convergence rate to fairness.

EMKC's problem of steady-state packet loss can be overcome by AVQ; however, the latter may exhibits *transient* overshoot before settling in its max-min fair stationary state. To understand this effect in detail, we repeat the simulation in topology $\mathcal{T}_1$ and increase α to 2 mb/s. As Fig. 28(a) shows, the instantaneous rate reaches 13 mb/s and the transient overshoot lasts for over 50 seconds. Moreover, this situation becomes even worse when the number of competing flows increases. As seen in Fig. 28(b), where 20 EMKC-AVQ users share the same 10-mb/s link in $\mathcal{T}_1$, the transient overshoot reaches 400% and lasts for tens of seconds. This situation is a consequence of the steady-state dynamics inherited from EMKC and the same term $N\alpha/\beta$ responsible for the overshoot, which is a linear function of the number of flows N and parameter α. This leads to a similar tradeoff between packet loss and convergence rate as in EMKC.

As mentioned in Section 3.3, RCP also suffers transient overshoot. This is be-

cause when a new flow joins the network, it simply sets its sending rate to the current rate $r_l(t)$ at the bottleneck link l. For a link that is already in its equilibrium (i.e., $y_l(t) = C_l$), this immediately leads to overflow of the link and a sudden surge in the queue size (relevant plots are omitted for brevity). In addition, the amount of overshoot is proportional to the number of arriving flows and its effect becomes progressively severe when many flows simultaneously join the system. To better see this, consider the following simulation, where a single link, whose capacity is 100 mb/s, RTT is 100 ms, and buffer size equals the bandwidth-delay product (i.e., 1.25 MB), is shared by 51 RCP flows with homogeneous RTT. For ease of reference, we denote this topology by $\mathcal{T}_5$. One flow starts first and the other fifty flows join after 30 seconds. We monitor at the bottleneck link the packet loss rate, which is calculated using the ratio between the numbers of dropped and received packets every link's control interval, i.e., $\min(d_l, 10 \text{ ms})$. As illustrated in Fig. 29(a), when the fifty flows arrive into the system, the combined incoming rate at the bottleneck immediately overflows the router buffer, resulting in transient packet loss as high as 98% and up to 66 MB dropped data. Thus, in highly dynamic scenarios, such as the Internet, where multiple flows frequently join and leave the network, RCP may experience significant packet loss (unless unrealistically large buffers are provisioned inside routers). This situation is demonstrated in the simulation given in Fig. 29(b), in which every 10 seconds ten RCP flows arrive at a 100-mb/s link and each flow has a random lifetime between 1 and 15 seconds. As seen from the figure, the bottleneck link suffers periodic packet loss high as 72%. This packet loss may further lead to drastic rate reductions,[5] retransmissions, slow convergence, and even instability.

[5]Note that RCP does not specify how flows react to packet loss or recover dropped packets ([26] uses very large buffers for all simulations to prevent packet loss). However, a common technique [56] is to use TCP's recovery mechanism (i.e., reducing rate in half) until all lost packets are recovered.

4 Max-Min Bottleneck Assignment

This section highlights the importance of analyzing discontinuous stability of max-min congestion control and explains some of the phenomena observed in the previous section.

4.1 General Stability Considerations

One of the most overlooked issues in the analysis of max-min feedback systems is instability arising from bottleneck oscillations and/or *inconsistent* bottleneck assignment (i.e., when flows incorrectly infer their bottlenecks). Analysis of max-min stability in multi-router networks is difficult (if not intractable) within the literature of modern congestion control as it involves non-linear systems that may switch between stationary points corresponding to different bottleneck assignments. Traditional switching theory [24] usually assumes that 1) the stationary point is preserved between the discontinuous jumps and 2) each subsystem corresponding to a *fixed* bottleneck assignment has only *one* stationary point. Under max-min feedback, both conditions may be violated since not only does each subsystem have a different stationary point, but it also may exhibit multiple equilibrium states or be unstable altogether.

Due to the complexity of the problem, the goal of this section is not to rigorously derive max-min stability of the existing methods, but to uncover the conditions that lead to instability and understand how to design stable max-min controllers in the future.

4.2 Why Bottleneck Assignment Is Important

We start with the following definition of bottleneck.

Definition 6 (Bertsekas-Gallager [9])**.** *A link is a bottleneck of flow r, if it is fully*

utilized and the rate of flow r is no less than that of any other flow accessing the link.

Under max-min feedback [56, 115], it is usually assumed that each flow x_r has a *fixed* bottleneck b_r, which does not change over time. It is further assumed that flows *not* bottlenecked by b_r do not contribute to feedback p_r generated by b_r. In multi-link topologies, this is certainly not the case since each flow x_s bottlenecked at some *other* link and passing through b_r clearly affects the value of p_r and thus the rate of flow x_r. If it also happens that x_r in turn affects x_s at bottleneck b_s, the system forms a closed loop that may become unstable. We study the formation of such loops in the context of MKC (Max-min Kelly Control) [115]; however, a similar question arises in other max-min feedback systems.

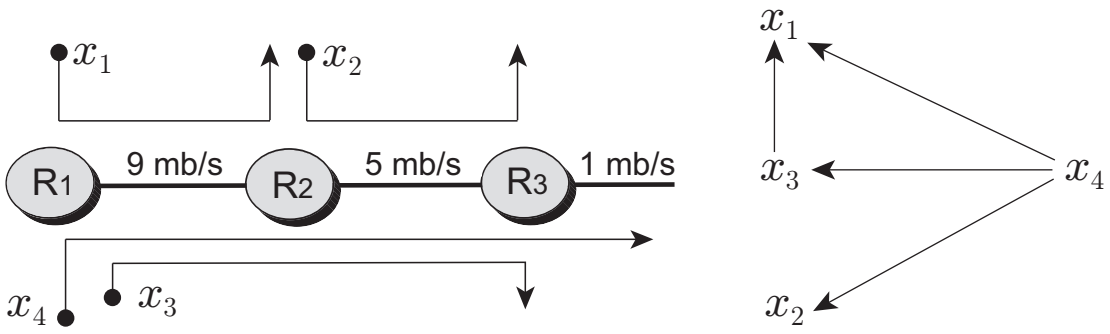
Assume that N users share M links in the network and suppose that $R \in \mathbb{R}^{N\times M}$ is the routing matrix of end-flows (i.e., $R_{rl} = 1$ if user r uses link l and 0 otherwise). Similarly, we define *bottleneck assignment* $B \in \mathbb{R}^{N\times M}$ of this multi-link system as an $N \times M$ matrix, where entry $B_{rl} = 1$ if user r is bottlenecked at link l and $B_{rl} = 0$ otherwise. Define b_r to be the bottleneck resource of user r and re-write the general form of MKC [115] as follows:

$$x_r(n) = \big(1 - \beta p_r(n - D^{\leftarrow}_{r,b_r})\big)x_r(n - D_r) + \alpha, \tag{131}$$

where

$$p_r(n) = p\Big(\sum_{s=1}^{N} R_{sb_r} x_s(n - D^{\rightarrow}_{s,b_r})\Big). \tag{132}$$

Notice that the sum in (132) includes the users bottlenecked by b_r (which we call *responsive* with respect to b_r), as well as any additional flows (which we call *unresponsive*) passing through the link. Even though each flow's feedback in (131)-(132) is still delayed by only *one* backward delay $D^{\leftarrow}_r = D^{\leftarrow}_{r,b_r}$, each flow s may affect other flows through as many as M forward delays $D^{\rightarrow}_{s,1}, \ldots, D^{\rightarrow}_{s,M}$. This presents



(a) topology (b) dependency graph

Fig. 30. Example that shows the effect of unresponsive flows.

a problem in stability analysis since the z-transform of the delay matrix and the Jacobian of the system are no longer block-diagonal and the proof in [115] does not hold.

Analysis below uses notation $x_s \to x_r$ to represent the fact that an unresponsive flow x_s passes through bottleneck b_r and affects flow x_r through feedback $p_r(n)$. For the example in Fig. 30(a) and max-min assignment of bottlenecks, we have $b_1 = 1, b_2 = b_3 = 2, b_4 = 3$ and the corresponding dependency graph is shown in Fig. 30(b).

Lemma 17. *For any system with max-min feedback that can stabilize its bottleneck assignment $b_1, \ldots, b_N$, the resulting dependency graph of (131)-(132) is acyclic.*

Proof. Suppose that the bottleneck assignment does not change over time and the dependency graph has a directed cycle $x_{i_1} \to \ldots \to x_{i_k} \to x_{i_1}$ for some $k \geq 2$. Notice that since flow x_{i_1} is unresponsive with respect to flow x_{i_2}, its stationary feedback $p^*_{i_1}$ must be larger than $p^*_{i_2}$ (otherwise, x_{i_1} would have switched its bottleneck to b_{i_2}). Generalizing this to the entire cycle, we immediately get a contradiction $p^*_{i_1} > p^*_{i_2} > \ldots > p^*_{i_k} > p^*_{i_1}$. Assuming a consistent tie-breaking rule obeyed by all flows, the above argument applies to cases where multiple links have equal steady-

state loss. □

Generalizing this lemma, we define a bottleneck assignment as *consistent* if it has an acyclic dependency graph. Then, we have the following result.

Lemma 18. *System (131)-(132) with a consistent bottleneck assignment* $b_1, \ldots, b_N$ *contains at least one router that has no unresponsive flows.*

Proof. Assume in contradiction that each link l has some unresponsive flow u_l passing through it and that this situation persists over time. Take the first unresponsive flow u_1 and notice that it is affected by some other unresponsive flow, which we label u_2, passing through u_1's bottleneck b_{u_1}. This leads to $u_1 \leftarrow u_2$. Repeating this reasoning for u_2, we get $u_1 \leftarrow u_2 \leftarrow u_3$, for some unresponsive flow u_3 at bottleneck b_{u_2}. This process continues and creates an infinite sequence $u_1 \leftarrow u_2 \leftarrow u_3 \leftarrow \ldots$ Since the number of unresponsive flows is finite, there is a point k when the sequence repeats itself (i.e., $u_k = u_j,\ j < k$) and we obtain a cycle in the dependency graph. □

Equipped with Lemmas 17 and 18, we next prove MKC's stability under any time-invariant bottleneck assignment.

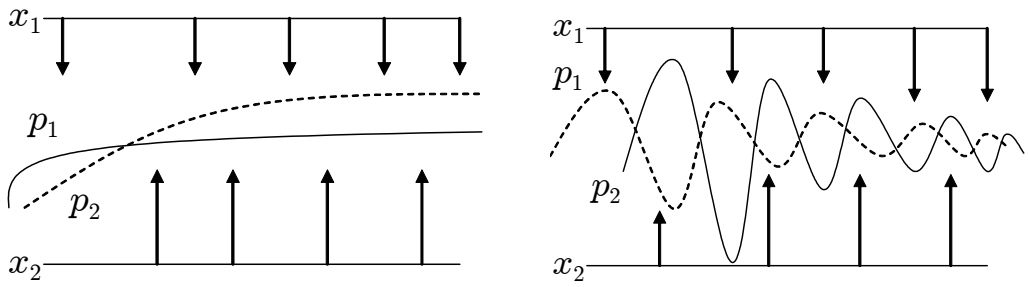
Theorem 14. *Under any bottleneck assignment* B *that does not change over time, MKC (131)-(132) is locally asymptotically stable regardless of delay if and only if for each link* l*, the subsystem composed of link* l *and flows* $\{x_r | B_{rl} = 1\}$ *is stable regardless of delay.*

Proof. Since bottlenecks do not shift and MKC relies on max-min feedback, Lemma 17 implies that the dependency graph is acyclic and bottleneck assignment is consistent. Using Lemma 18, there exists at least one link l_1 with no unresponsive flows. Then, it follows that all flows passing through l_1 are bottlenecked by l_1 and their stability is independent of the dynamics of the remaining flows. After the users bottlenecked

by l_1 converge to their stationary rates, we can remove l_1 and all of its (constant-rate) flows from the system. The new network still exhibits max-min bottleneck assignment and thus contains some link l_2 that has no unresponsive flows. Repeating this argument for all links $l_1, \ldots, l_M$, we obtain that the local dynamics of the entire system can be viewed as a system of linear block-diagonal equations with matrix $A = \text{diag}(A_1, \ldots, A_M)$, where $A_l \in \mathbb{R}^{N_l \times N_l}$ is the Jacobian matrix of N_l flows bottlenecked at link l ($\sum_{l=1}^{M} N_l = N$). Thus, we arrive at the conclusion that the entire system achieves delay-independent stability if and only if the individual bottlenecks do. □

While the general issue of bottleneck oscillation still remains open, this section shows that as long as flows can properly select their most-congested links and avoid dependency cycles, the dynamics of multi-link systems are in fact described by those of individual links. Also notice that if flows converge their feedback *monotonically* for any bottleneck assignment, all cycles in the dependency graph are self-correcting (i.e., they eventually lead to a contradiction similar to the one in Lemma 17). This is schematically shown in Fig. 31(a), where two flows x_1 and x_2 sample monotonic feedback p_1 and p_2 from two links common to both flows. While their initial inference of bottlenecks may be inconsistent, the situation is eventually self-correcting and both flows agree that feedback p_2 should be applied to their equations.

On the other hand, when feedback oscillates there is a possibility of having a directed cycle $x_{i_1} \to \ldots \to x_{i_k} \to x_{i_1}$ that persists over time. This can be shown using the example of two flows. Suppose cycle $x_1 \to x_2 \to x_1$ exists and is not self-correcting. This implies that flow x_2 affects x_1 at bottleneck b_1 and x_1 affects x_2 at link b_2. Since the two flows sample feedback p_1 and p_2 from their respective bottlenecks at *different* times, the apparent contradiction $p_1 > p_2 > p_1$ is actually a perfectly legitimate set of *two* independent conditions: $p_1(n_1) > p_2(n_1)$ and $p_2(n_2) > p_1(n_2)$ for



(a) monotonic (b) non-monotonic

Fig. 31. Types of multi-router feedback.

some time instants $n_1 \neq n_2$. Therefore, as long as p_1 and p_2 oscillate, it is possible that x_1 at time n_1 infers that $p_1 > p_2$, while x_2 at time n_2 infers the opposite (i.e., $p_2 > p_1$). An example of this is illustrated in Fig. 31(b), where both p_1 and p_2 are individually (i.e., without the max function) stable, but create a cyclic dependency graph with potential for instability.

As the XCP examples show, non-monotonic feedback allows flows to continuously switch between bottlenecks and maintain persistent cycles in the dependency graph, which eventually leads to instability. It thus becomes imperative that flows correctly choose their bottlenecks, which is what EMKC achieves in practice due to its more predictable (i.e., monotonic) evolution of feedback at each link. We summarize the conclusion of this section in the following corollary.

Corollary 7. *Max-min congestion control that converges its feedback $p_l(n)$ at each link l monotonically to some stationary point, regardless of the bottleneck assignment, is stable over multi-link topologies if and only if the corresponding bottlenecks are.*

Note that EMKC in general does not satisfy this requirement (i.e., there are delay patterns that create small disturbances to the ideal convergence behavior); however, out of the studied methods, it has the best control over delay and exhibits dynamics that can be deemed monotonic in many practical cases.

5 JetMax

In this section, we present JetMax and provide an analytical study of its properties. The next section discusses implementation and performance details of this protocol.

5.1 Design

Consider link l at time n. Assume that $N_l(n)$ is the number of *responsive* flows in this router at time n and $w_l(n)$ is their combined rate. Also, assume that $u_l(n) = y_l(n) - w_l(n)$ is the aggregate rate of *unresponsive* flows at the router and $0 < \gamma_l \leq 1$ is its desired utilization level. The main idea of JetMax is to equally divide the residual bandwidth $\gamma_l C_l - u_l(n)$ between all flows bottlenecked by the router and then provide this average rate to all responsive users. Knowing $u_l(n)$ and $N_l(n)$ (methods of computing these are discussed later in Sections 6.1 and 6.2), the router periodically (i.e., every Δ_l time units) calculates and feeds back to the senders the fair rate $g_l(n)$:

$$g_l(n) = \frac{\gamma_l C_l - u_l(n)}{N_l(n)}. \tag{133}$$

which is later utilized by end-users in their control equations:

$$x_r(n) = (1-\tau)x_r(n - D_r) + \tau g_l(n - D_r^{\leftarrow}), \tag{134}$$

where constant $\tau > 0$. Clearly, $x_r(n)$ is in fact an exponential weighted moving average of $g_l(n)$ with weight τ. An alternative interpretation of (134) can be obtained by rewriting it as $x_r(n) = x_r(n - D_r) - \tau(x_r(n - D_r) - g_l(n - D_r^{\leftarrow}))$. In this view, equation (134) is actually an Integral controller of signal $x_r(n)$ with a time-varying set point $g_l(n - D_r^{\leftarrow})$. Note that utilizing classical PID control theory, Blanchini *et al.* [14] proposed another method that is robust to delay. However, these two schemes are developed based on different theoretical foundations. In addition, in contrast to

this method, JetMax does not monitor queue length in the router or estimate the maximum RTT to achieve stability.

Besides the end-user equation, another important issue is the bottleneck switching mechanism. A straightforward solution is that each user chooses the link along its path with the *smallest* $g_l(n)$ as the bottleneck resource. However, as the bottleneck assignment shifts (i.e., flows migrate from one link to another), both $N_l(n)$ and $u_l(n)$ change accordingly. Thus, the value of $g_l(n)$ experiences sudden changes, making the system susceptible to transient oscillations during bottleneck switchings. We also observe this phenomenon in simulations and omit the corresponding plots for brevity. This issue can be overcome by replacing $g_l(n)$ with packet loss rate $p_l(n)$, which is a function of the combined ingress rate $y_l(n) = w_l(n) + u_l(n)$, i.e.,

$$p_l(n) = \frac{y_l(n) - \gamma_l C_l}{y_l(n)}. \tag{135}$$

Then, the bottleneck link of a given flow is the one with the *largest* $p_l(n)$ in the path. Since $y_l(n)$ remains the same immediately after a bottleneck shift and so does $p_l(n)$, JetMax, as shown in both `ns2` simulations (Section 7) and Linux experiments (Section 8), exhibits smooth transition during bottleneck switching. We note that JetMax is a combined framework, which employs a rate-based scheme at end-users to adjust their sending rates and queue-based method inside routers to decide the bottleneck router. We refer interested readers to [23] for an in-depth discussion of the relationship between rate- and queue-based congestion controls.

In the rest of this section, we prove JetMax's delay-independent stability, max-min fairness in the steady state, and ideal convergence speed to stationarity.

5.2 Delay-Independent Stability

We start by deriving the stationary rate of each flow.

Lemma 19. *Given that flow r is bottlenecked by a resource l of capacity C_l together with $N_l(n)-1$ other flows, its stationary sending rate is $x_r^* = (\gamma_l C_l - u_l^*)/N_l^*$, where u_l^* and N_l^* are the steady-state values of $u_l(n)$ and $N_l(n)$ at link l.*

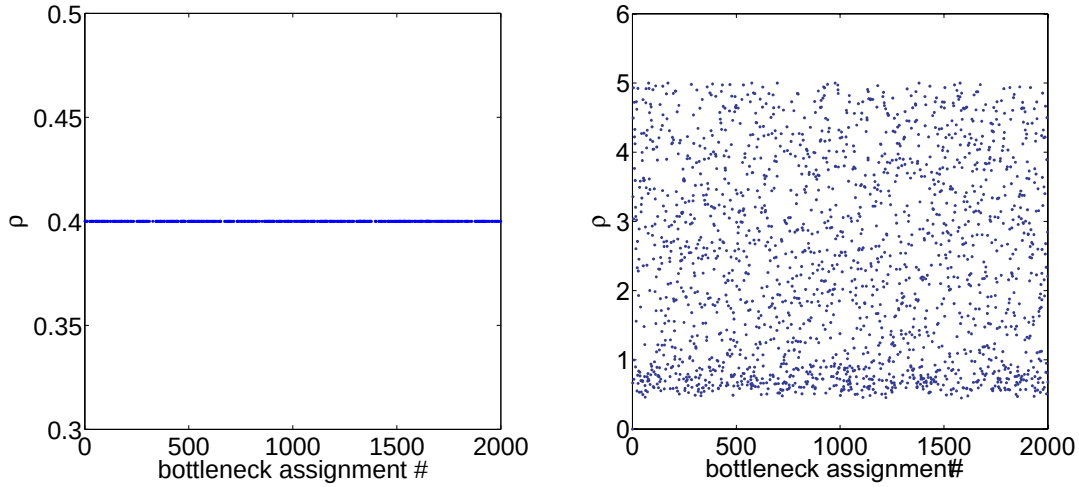
In the steady state, we have $x_r(n) = x_r(n-D_r) = x_r^*$ and $u_l(n) = u_l^*$. Combining this with JetMax's end-user equation (134) immediately yields $x_r^* = (\gamma_l C_l - u_l^*)/N_l^*$.

We next show that, under any consistent bottleneck assignment, stability analysis of system (133)-(134) can be reduced to that of EMKC.

Theorem 15. *Under any consistent bottleneck assignment, JetMax (133)-(134) is stable regardless of delay if and only if $0 < \tau < 2$.*

Proof. First, consider an undelayed JetMax system with a single link l. Since the bottleneck assignment is given, $N_l(n)$ is fixed, i.e., $N_l(n) = N_l^*$. Then, Jacobian matrix A_l of the subsystem corresponding to link l is simply $A_l = \mathrm{diag}(1-\tau)$, which is stable if and only if $\rho(A_l) = |1-\tau| < 1$, or in other words, $0 < \tau < 2$. Next, combining the fact that A_l is symmetric and using Theorem 1 in [115], we obtain that single-link JetMax is stable for all types of directional and time-varying delay under the same condition on τ. Finally, invoking Theorem 14, we arrive at the conclusion that JetMax achieves delay-independent stability in any multi-link network with a consistent bottleneck assignment if and only if its individual bottlenecks do, i.e., $0 < \tau < 2$. □

It is worth noting that in the above proof and the following analysis of JetMax's convergence properties, $N_l(n)$ is assumed to be known to the router. This assumption is realized by the estimation technique described in Section 6.1. As demonstrated later in the chapter, this proposed method is very accurate in both `ns2` simulations and Linux experiments.



(a) consistent (b) inconsistent

Fig. 32. Spectral radius $\rho(A)$ of system (133)-(134) with $\tau = 0.6$ under 2000 random bottleneck assignments.

To better understand Theorem 15, we set $\tau = 0.6$ and generate 2000 random bottleneck assignments in random topologies with 10 routers and 50 flows. For each case, we decide whether the topology is consistent or not by applying DFS (depth-first search) to the corresponding dependency graph. As predicted by Theorem 15, the system under any consistent bottleneck assignment is stable and has a spectral radius $\rho(A) = |1 - \tau| = 0.4$, which perfectly aligns with the simulation result illustrated in Fig. 32(a). At the same time, as Fig. 32(b) demonstrates, $\rho(A)$ under inconsistent bottleneck assignments may exceed 1, in which case even the undelayed system is unstable.

5.3 Max-Min Fairness

From Lemma 19, notice that the stationary packet loss p_l^* of all congested links is zero. Thus, if there are multiple links with zero packet loss in the path of a flow r, it will be uncertain which link should be chosen such that the resulting bottleneck

assignment is max-min fair. To deal with this situation, we introduce a simple tie-breaking rule based on the average rate of the responsive flows at each link. Assuming that several links tie in *zero* packet loss, the user prefers the link with the smallest value of $g_l = (\gamma_l C_l - u_l)/N_l$, i.e., it sets

$$b_r = \arg \min_{l \in r: p_l^* = 0} g_l(n). \tag{136}$$

To maintain stability, switching based on the largest packet loss (135) may be performed at any time n; however, that based on (136) is conducted only when flow r's sending rate reaches the ε-neighborhood of stationarity under the current bottleneck assignment. We next prove max-min fairness of the resulting system.

Theorem 16. *The stationary resource allocation of JetMax (134)-(136) is max-min fair.*

Proof. Suppose in contradiction that JetMax is not max-min fair in its steady state. Then, using max-min results in Bertsekas-Gallager [9, pp. 527], there must exist flow r that is not bottlenecked by any link in its path. Let $l \in r$ be the link that provides feedback to flow r. Then, from Lemma 19 we must have that link l is fully utilized and stationary rate $x_r^* = (\gamma_l C_l - u_l^*)/N_l^*$. According to Definition 6, flow r is not bottlenecked by this link if and only if there exists a flow s accessing l such that

$$x_r^* < x_s^*. \tag{137}$$

Let flow s be constrained by link k where $k \neq l$. Then, we have $x_s^* = (\gamma_k C_k - u_k^*)/N_k^*$, which translates (137) into

$$\frac{\gamma_l C_l - u_l^*}{N_l^*} < \frac{\gamma_k C_k - u_k^*}{N_k^*}. \tag{138}$$

According to (136), however, the last inequality must force the bottleneck of

flow s to shift from link k to l, thus contradicting the assumption that the system has reached stationarity. □

5.4 Capacity-Independent Convergence Rate

For the analysis of convergence rate, we focus on *single-link* behavior of JetMax as it generally serves as a good indicator of multi-link performance of this method. To formalize the metric "convergence rate," consider the following definition.

Definition 7. *A protocol converges to* $(1-\varepsilon)$*-efficiency in* n_e *steps if the system starts with* $y(0)=0$ *and* n_e *is the smallest integer satisfying*

$$\forall n \geq n_e : \ \frac{y(n)}{\gamma C} \geq 1-\varepsilon \tag{139}$$

Similarly, $(1-\varepsilon)$*-fairness is reached in* n_f *steps if the system starts in the maximally unfair state (i.e.,* $\exists r, x_r(0)=\gamma C$ *and* $\forall i \neq r, x_i(0)=0$*) and* n_f *is the smallest integer satisfying*

$$\forall n \geq n_f : \frac{|x_r(n)-x_r^*|}{x_r^*} \leq \varepsilon, \ \forall r. \tag{140}$$

The following result derives capacity-independent convergence time of JetMax.

Theorem 17. *On a single link, JetMax reaches both* $(1-\varepsilon)$*-efficiency and* $(1-\varepsilon)$*-fairness in* $\lceil \log_{|1-\tau|} \varepsilon \rceil$ *RTTs.*

Proof. Without loss of generality, assume homogeneous feedback delay for each flow, consider any consistent bottleneck assignment, and focus on link l. Next, combine the sending rate (134) of all flows bottlenecked by l into the aggregate rate $y_l(n) = \sum_{r \in l} x_r(n)$. Solving the resulting recurrence on $y_l(n)$, we obtain that the combined rate at time n can be written as

$$y(n) = (1-\tau)^{n/D}(y(0)-\gamma C)+\gamma C, \tag{141}$$

where D is the RTT of end-flows and $y(0) = 0$ is the initial total rate of all flows. Combining the last equation with (139) and writing n_e in terms of RTT steps, we get $|1-\tau|^{n_e} \leq \varepsilon$, which yields

$$n_e = \lceil \log_{|1-\tau|} \varepsilon \rceil. \tag{142}$$

Next, assume that the system starts in the maximally unfair state (i.e., one flow takes all bandwidth) and that unresponsive flows are stabilized. Therefore, controller (134) becomes

$$x_r(n) = (1-\tau)x_r(n-D_r) - \tau x_r(n-D_r). \tag{143}$$

Solving this recurrence, we get

$$x_r(n) = (1-\tau)^{n/D_r}(x(0) - x_r^*) + x_r^*, \tag{144}$$

which shrinks to $(1-\varepsilon)$-fairness in $n_f = \lceil \log_{|1-\tau|} \varepsilon \rceil$ RTT steps following the technique we used to obtain (142). □

This theorem indicates that JetMax reaches full utilization and converges to fairness over links of *any* capacity in the same number of steps (verification of this result using simulations and experiments follows later). Also observe from (141) and (144) that $0 < \tau < 1$ is required to guarantee monotonicity of the controller. Thus, all JetMax experiments in this chapter use $\tau = 0.6$ unless otherwise specified.

Next, we provide implementation details of JetMax and evaluate its performance via both `ns2` simulations and Linux experiments.

6 Implementation

6.1 Estimating Number of Flows

The first issue encountered by a JetMax router l is how to estimate the current number of responsive flows $N_l(n)$. Dynamic tracking of active flow population $N_l(n)$ has been actively studied in ATM networks [6, 36, 104]. Specifically, as suggested in [6, 36], $N_l(n)$ can be simply approximated by the ratio between $w_l(n)$ and $g_l(n)$. However, similar to RCP discussed in Section 3.8, this method may result in significant transient overshoot of the bottleneck link when new flows join the system. Another scheme introduced in [104] is in spirit similar to our method presented below. However, it assumes a constant cell (packet) size for all connections and is not suitable for the current Internet where packet sizes may be different between flows and over time.

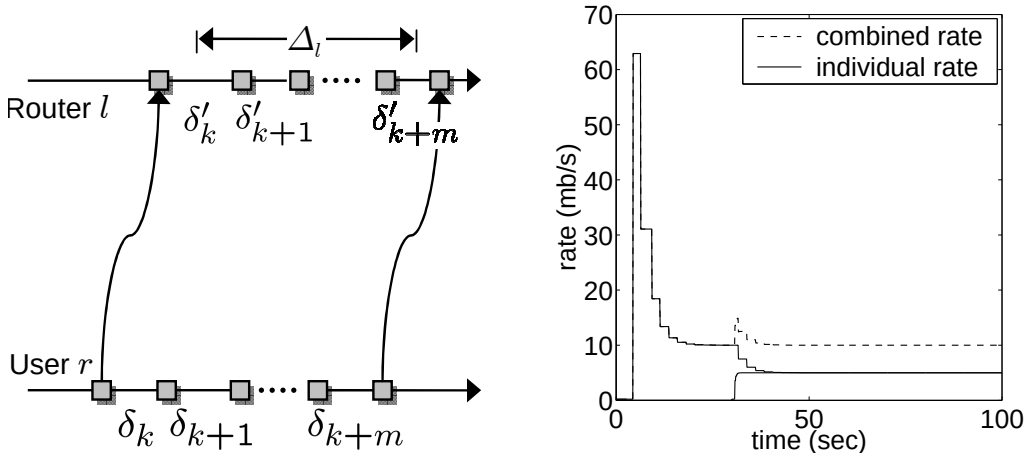
Our solution to this problem is based on the following observations. For a given flow r, assume that δ_k is the inter-packet departure delay between packets k and $k+1$ at the source and δ'_k is the corresponding inter-packet arrival delay at link l. Fig. 33(a) illustrates this notation and shows that the router's control interval Δ_l generally starts and ends in-between two arriving packets. We therefore have the following relationship between the router's control interval and the combined delay of all packets from flow r observed during the interval

$$\sum_{i=k+1}^{k+m-1} \delta'_i \leq \Delta_l \leq \sum_{i=k}^{k+m} \delta'_i, \tag{145}$$

where $k+m$ is the packet that arrives immediately after the end of this interval. This further yields

$$\lim_{\Delta_l \to \infty} \frac{\sum_{i=k}^{k+m} \delta'_i}{\Delta_l} = \lim_{\Delta_l \to \infty} \frac{\sum_{i=k+1}^{k+m-1} \delta'_i}{\Delta_l} = 1. \tag{146}$$

Generalizing this relation to all N_l flows bottlenecked by l and taking the sum-



(a) (b)

Fig. 33. (a) The relationship between control interval Δ_l and inter-packet interval δ_k; (b) JetMax ($\tau = 0.6$ and $\gamma = 1$) with the naive bottleneck switching scheme in $\mathcal{T}_1$.

mation of inter-packet delays over all such flows, we have

$$\lim_{\Delta_l \to \infty} \frac{\sum_{r=1}^{N_l} \sum_{i=k}^{k+m} \delta'_i}{\Delta_l} = N_l. \tag{147}$$

Even though in general δ_k does not equal to δ'_k, sums of these two metrics over a large number of packets are asymptotically equal, i.e., $\lim_{m\to\infty} \sum_{i=k}^{k+m} \delta_i = \lim_{m\to\infty} \sum_{i=k}^{k+m} \delta'_i$. This, combined with (147), leads to

$$\lim_{\Delta_l \to \infty} \frac{\sum_{r=1}^{N_l} \sum_{i=k}^{k+m} \delta_i}{\Delta_l} = N_l. \tag{148}$$

Using the last equation, we next develop a mechanism for estimating N_l. Each user r includes in every packet k its inter-packet departure delay $\delta_k = s_k/x_r(n)$, where s_k is the size of the packet and $x_r(n)$ is the current sending rate. The router then sums up this field over all packets of all *responsive* flows and averages this value over interval Δ_l. From (148), we have that the value $\tilde{N}_l = \sum_{r=1}^{N_l} \sum_{i=0}^{m} \delta_{k+i}/\Delta_l$ converges to the true number of flows N_l as Δ_l grows to infinity. Note that this method does

not maintain state information about individual flows and requires only one addition per arriving packet and one division per interval Δ_l.

We finally remark that choosing a large interval Δ_l improves accuracy of estimating N_l, but may reduce the router's responsiveness to dynamics of the incoming traffic. In practice, the above scheme works very well with small Δ_l (say, 100 ms). Thus, throughout the chapter we set $\Delta_l = 100$ ms unless otherwise specified. As demonstrated later via `ns2` simulations and Linux experiments, this mechanism is very effective and delivers accurate estimations of N_l in diverse scenarios (including those with "mice" traffic and random packet loss).

6.2 Maintaining Membership of Flows

JetMax relies on the existence of an effective mechanism for the routers to identify its responsive flows. To implement this functionality, we allocate three one-byte router-ID fields in the packet header: R_T, R_C, and R_S. All IDs are in terms of hop count from the source. The first field R_T records the router ID of the *true* (i.e., currently known to the source) bottleneck link b_r for a given flow r; the second field carries the hop number of the packet (which we call the *current* router-ID) and is incremented by each router; and the last field contains the *suggested* resource ID that is modified by the routers that perceive their congestion to be higher than that experienced by the flow at the preceding routers.

Upon each packet arrival, link l increments R_C by one and then examines its local packet loss $p_l(n)$ and the one carried in the packet. If both packet-loss values are zero, the router checks if its local average rate $g_l(n)$ is less than the one carried in the header. If either case is true, the router overwrites the packet loss and average rate in the packet header and additionally sets the packet's field R_S to the value of R_C obtained from the header. At the sending side, if the suggested router R_S carried

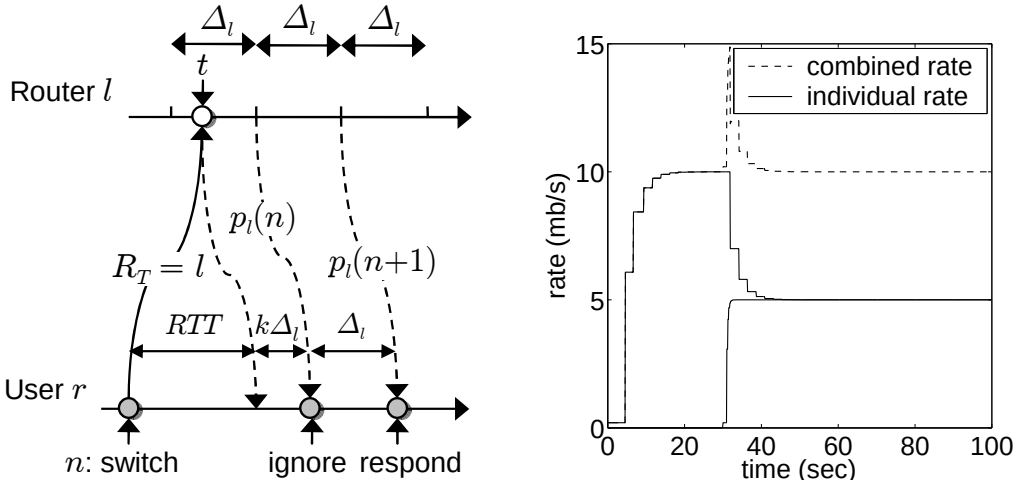
in the acknowledgment is different from the true router R_T, the source notices that a bottleneck switch is suggested and initiates a switch to R_S.

Calculations of responsive rate $w_l(n)$ and unresponsive rate $u_l(n)$ at router l can also be easily implemented. At the beginning of each control interval Δ_l, router l resets $w_l(n)$ and $u_l(n)$ to zero. For each incoming packets, the router, after incrementing R_C by one, checks whether R_C and R_T are equal. If they are, the packet is counted as responsive and its size s is added to $w_l(n)$; otherwise, the packet is considered unresponsive and its size is added to $u_l(n)$. Then, at the end of interval Δ_l, the router obtains the responsive and unresponsive rates by normalizing $w_l(n)$ and $u_l(n)$ by Δ_l. Clearly, the combined incoming rate $y_l(n)$ is simply $w_l(n) + u_l(n)$.

6.3 Managing Bottleneck Switching

The above scheme in itself is insufficient to eliminate all undesirable transient effects associated with bottleneck switching. To demonstrate this, we simulate the algorithms developed so far in `ns2` using the single-link topology $\mathcal{T}_1$, where we change the join order of users to highlight some of the issues arising in the naive implementation of JetMax. Specifically, flows x_2 and x_1 join at time 0 and 30 seconds, and experience round-trip delay 2020 and 220 ms, respectively. The simulation result is plotted in Fig. 33(b), in which x_2 initially overflows the link's capacity by 500% and then maintains non-zero packet loss for over 15 seconds. As we discuss below, this phenomenon arises as the result of improper management of bottleneck switching.

For the illustration in Fig. 34(a) that explains this situation, assume that user r changes its bottleneck to link l at time n and the first packet carrying this new membership arrives into link l at time $t = n + D^{\rightarrow}_{r,l}$, which is in the middle of the router's control interval Δ_l. Notice that flow r is counted as *unresponsive* prior to time t and *responsive* after that. This inconsistent inference of membership results in



(a) (b)

Fig. 34. (a) The scenario where the bottleneck switching occurs in the middle of the router's control interval; (b) JetMax ($\tau = 0.6$ and $\gamma = 1$) with the proper bottleneck switching scheme in $\mathcal{T}_1$.

an incorrect estimation of both N_l and $u_l(n)$. Consequently, the resulting feedback does not reflect the actual situation inside the router and leads to oscillations in the transient phase.

Fortunately, this inconsistency exists only in the *first* interval Δ_l after the switch. Thus, to properly manage bottleneck switching, the end-user simply ignores the first non-duplicate ACK after each switch and reacts to the following ones as shown in Fig. 34(a). We can also see from the figure that, using this mechanism, the end-user delays its response to ACKs by 1 RTT and $1 + k$ (where $0 \leq k \leq 1$) Δ_l after each switching. Simulation of the resulting JetMax is illustrated in Fig. 34(b), in which the initial "spike" present in Fig. 33(b) is eliminated and x_2 monotonically converges to efficiency. However, notice in the figure that JetMax exhibits transient packet loss reaching as high as 33% when flow x_1 joins the network. We explain and resolve this issue in the next subsection.

6.4 Eliminating Transient Packet Loss

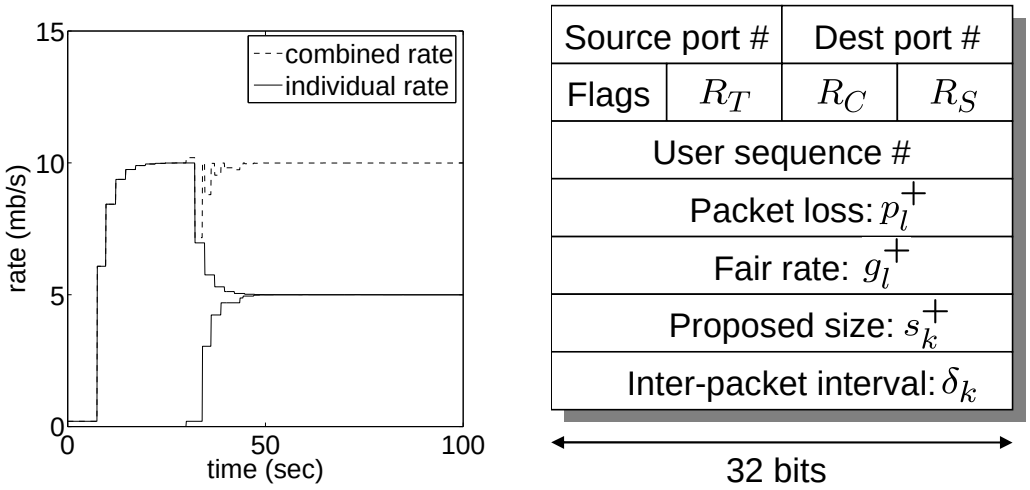
The reason of the transient packet loss shown in Fig. 34(b) lies in the fact that flow x_2 with a large RTT does not release bandwidth quickly enough and is not aware of the presence of any competing flows until after the overshoot has happened.

Proper implementation of JetMax that avoids this issue relies on the concept of "proposed rate." Suppose a JetMax flow decides to increase its sending rate; however, it does not know if the other flows in the system have released (or are planning to release) enough bandwidth for this increase not to cause packet loss. To resolve this uncertainty, the flow that plans to *increase* its rate first "proposes" the new rate in its packet header and waits for the router's approval/rejection decision based on the aggregate proposed rate at the router. Flows not interested in rate increase continuously propose their current sending rates and ignore the decisions they may be receiving. Furthermore, flows planning to *decrease* their rates can do so immediately as such actions can only reduce the traffic at the bottleneck and improve the fairness of the system.

This strategy can be easily realized in practice. Assuming that the k-th packet transmitted by the source has packet size s_k bits, the flow can convey its proposed rate $x_r^+(n)$ to the router by including a *virtual* packet size s_k^+ in each header such that

$$s_k^+ = s_k \frac{x_r^+(n)}{x_r(n)}. \tag{149}$$

For each incoming packet during interval Δ_l, the router increments $w_l^+(n)$ and $u_l^+(n)$ by virtual packet size s_k^+ based on the membership of the packet. At the end of each interval, the router normalizes $w_l^+(n)$ and $u_l^+(n)$ by interval duration Δ_l and gets the responsive and unresponsive proposed rates. The combined proposed rate $y_l^+(n) = w_l^+(n) + u_l^+(n) = \sum_{r \in l} x_r^+(n)$. Then, the router accepts $y_l^+(n)$ if it is less



(a) (b)

Fig. 35. (a) JetMax ($\tau = 0.6$ and $\gamma = 1$) with proposed rate in $\mathcal{T}_1$; (b) format of the JetMax packet header.

than $\gamma_l C_l$ and decline it otherwise. Note that when computing $g_l(n)$ in (133) and $p_l(n)$ in (135) at the end of each control interval, the router simply replaces $u_l(n)$ and $y_l(n)$ with their corresponding proposed values $u_l^+(n)$ and $y_l^+(n)$:

$$\begin{aligned} p_l^+(n) &= 1 - \frac{\gamma_l C_l}{y_l^+(n)} \\ g_l^+(n) &= \frac{\gamma_l C_l - u_l^+(n)}{N_l}. \end{aligned} \tag{150}$$

Clearly, no extra latency is introduced by this mechanism and each approved rate adjustment takes exactly one RTT (instead of two RTTs if (133)-(135) were based on actual rates). The result of this implementation is shown in Fig. 35(a), in which the system never overflows the link and converges to fairness monotonically.

6.5 Calculating Reference Rate

Intuitively, when applying control equation (134), the end-user can directly use the most recently proposed rate $x_r^+(n - D_r)$ (if approved by the router) as the next *actual*

rate $x_r(n)$ and apply $x_r^+(n - D_r)$ to computation of the next proposed rate $x_r^+(n)$. However, this may incur problems when bottleneck switching occurs in the middle of the bottleneck router's control interval. In this case, the average incoming rate computed by the router is a function of previous and current proposed rates. As a consequence, the router may erroneously approve a proposed rate that is actually above the link's capacity or reject one even when the link is under-utilized, both of which may further lead to transient rate, or even bottleneck, oscillations. Leveraging the fact that this inconsistency exists only in the *first* control interval after the switch, we solve this problem by letting the end-user ignore the first non-duplicate ACK after the switch and respond to the remaining ones. Also note that, analogous to the discussion in Section 6.3, time for each rate adjustment becomes $RTT + (1 + k)\Delta_l$ where $0 \leq k \leq 1$.

6.6 Packet Format

The header format of a JetMax packet is illustrated in Fig. 35(b). Besides the two-byte fields for port numbers, we allocate a one-byte field to each of *flags*, R_T, R_C, and R_S. Then, we use four-byte numbers to record the user sequence numbers to deal with out-of-order packets, packet loss p_l^+, fair rate g_l^+, user-proposed packet size s_k^+, and the inter-packet interval $\delta_k = s_k/x_r(n)$. Note that only δ_k uses the actual sending rate of the flow.

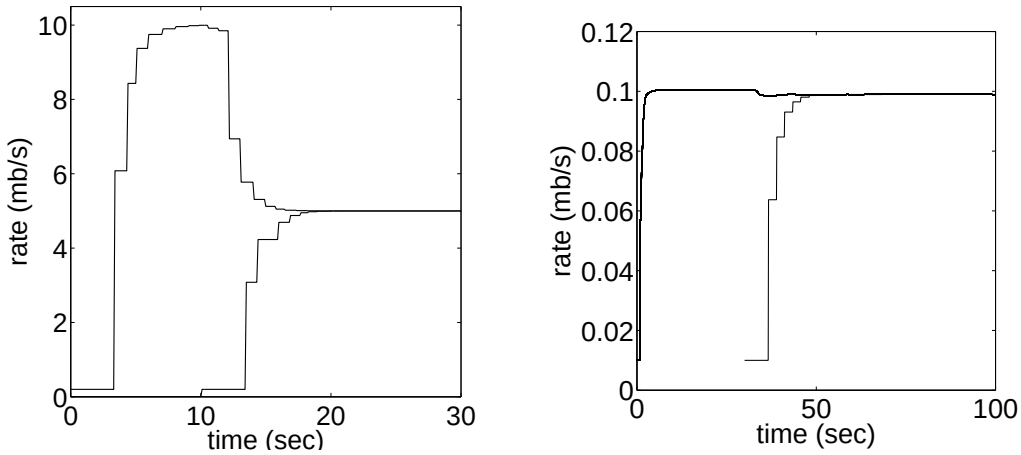
Thus, the total size of a JetMax packet header is 28 bytes, which is 4 bytes smaller than XCP's 32 (12 XCP-specific bytes and 20 bytes of the TCP header). In addition, JetMax's per-packet processing inside the router takes only three additions for responsive flows (to calculate R_C, w_l^+, and N_l) and two additions for unresponsive flows (to compute R_C and u_l^+), as opposed to XCP's three multiplications and six additions [56].

7 Simulations

7.1 Behavior in $\mathcal{T}_2$, $\mathcal{T}_3$, $\mathcal{T}_4$, and $\mathcal{T}_5$

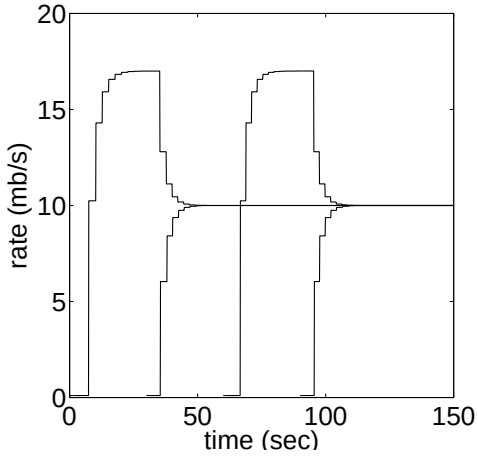
We first repeat the `ns2` simulations that earlier presented stability and equilibrium problems to existing methods and then examine how JetMax handles additional scenarios. Simulation code used in this work is available in [48].

Performance of JetMax in $\mathcal{T}_2$, $\mathcal{T}_3$, $\mathcal{T}_4$, and $\mathcal{T}_5$ is shown in Fig. 36, in which the protocol demonstrates monotonic convergence, max-min allocation of bottleneck resource in the equilibrium, effective handling of bottleneck selection, and loss-free operation in both the transient phase and steady state. Numerical data from the simulations also show that the system never overshoots the link's capacity or loses any packets. Simulations in a dozen of additional (more complex) multi-link topologies combined with both fixed and random feedback delay produce similar results and are omitted for brevity. It is also worthwhile to note that the flat regions in Fig. 36(a) when both flows start consume three RTTs (i.e., 2.6 seconds) and are necessary for the flows to deal with initial router assignment and bottleneck selection.

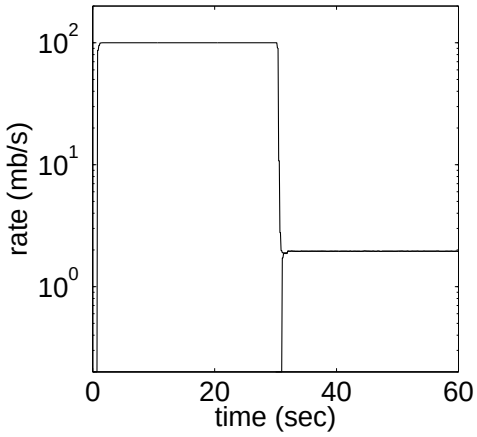


(a) $\mathcal{T}_2$

(b) $\mathcal{T}_3$



(c) $\mathcal{T}_4$



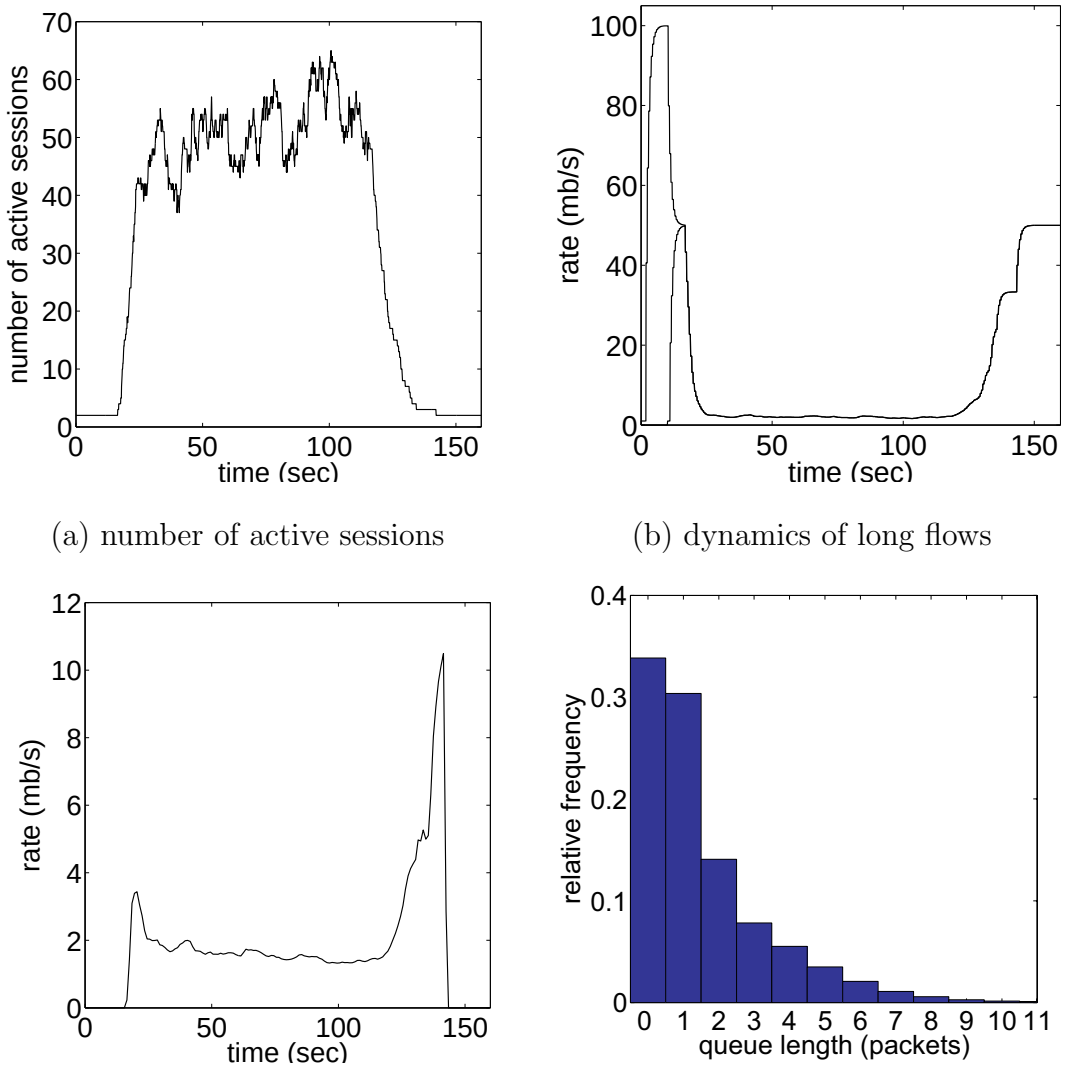
(d) $\mathcal{T}_5$

Fig. 36. Performance of JetMax ($\tau = 0.6$ and $\gamma = 1$) in `ns2`.

7.2 Effect of Mice Traffic

All of our simulations so far have been performed in environments with long-lived flows. However, the real Internet traffic is composed of a mixture of connections with a wide range of transfer sizes, packet sizes, and RTTs [34]. Thus, to obtain a better understanding of JetMax, we next test it in more diverse scenarios.

Toward this end, we first consider a simple "dumb bell" topology, where 2 long and 500 short JetMax flows share a single link of capacity 100 mb/s.The inter-arrival time of short flows follows an exponential distribution with mean $\lambda = 0.2$ seconds and the duration of each flow is drawn from a log-normal distribution [83] with mean $\omega = 10$ seconds. From basic queuing theory, we can infer that the expected number of active short flows at any instant is $L = \omega/\lambda = 50$, while the instantaneous flow population is bursty as illustrated in Fig. 37(a). Moreover, we set the packet sizes of the short flows to be uniformly distributed in $[800, 1300]$ bytes and their RTTs are selected uniformly randomly in $[40, 1040]$ ms.



(a) number of active sessions

(b) dynamics of long flows

(c) average rate of short flows

(d) queue length distribution

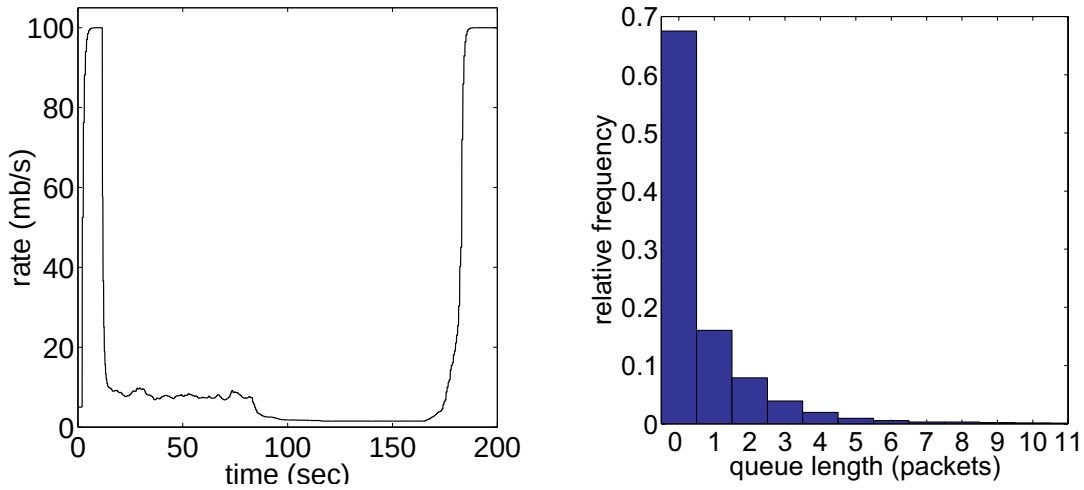
Fig. 37. Single-link performance of JetMax ($\tau = 0.6$ and $\gamma = 1$) in the presence of mice flows.

As seen in Fig. 37(b), one long flow starts first and quickly reaches link utilization. After the second long flow joins 5 seconds later, the first flow is forced to release some of its bandwidth, allowing both flows to converge to the fair share of the link's capacity (i.e., 50 mb/s). At time 15 seconds, mice flows start joining and leaving the network. Since on average there are 50 short and 2 long flows in the system, the *expected* fair rate is $100/52 = 1.92$ mb/s per flow. This prediction is confirmed in Fig. 37(b), where the sending rates of the long flows remain within $[1.7, 2.0]$ mb/s during the period between $[30, 120]$ seconds. It is worth noting that the small rate oscillations during this interval are not due to instability, but the time-varying number of mice flows and changes to the stationary point of the system.

To understand the throughput obtained by the short flows, Fig. 37(c) shows the average rate of mice traffic. As seen in the figure, the short flows also manage to obtain their fair share (despite the short duration) and achieve rates close to the expected 1.92 mb/s. This also confirms the effectiveness of the JetMax router in estimating the number of locally congested flows N_l. As the number of active connections decreases after time 120 seconds, sending rates of the remaining short flows climb up and take over the bandwidth of the departed flows.

We also plot the queue length distribution of the bottleneck link in Fig. 37(d), in which we sample the instantaneous queue size every 10 ms. The bin size of the histogram is one packet. As can be seen from the plot, for 64% of the time the queue has less than two packets and 99% of the time less than 10 packets. Thus, JetMax is successful in maintaining small buffers and, as a consequence, does not lose any packets or increase queuing delays in practice.

We next test JetMax's multi-link performance in the presence of mice flows. Consider a "parking lot" topology where a long flow traverses two links R_1 and R_2 of capacities 400 and 100 mb/s, each of which is accessed by 500 short flows. In addition,



(a) dynamics of the long flow (b) queue length distribution of R_1

Fig. 38. Multi-link performance of a long JetMax flow ($\tau = 0.6$ and $\gamma = 1$) in the presence of mice traffic.

we set Δ_l to be uniformly random in $[100, 300]$ ms to test JetMax's performance when routers have heterogeneous control intervals. As shown in Fig. 38(b), the long flow starts first and converges to the capacity of R_2. Short flows accessing R_1 start joining the system after 15 seconds. Since R_1 becomes more congested than R_2, the long flow switches the bottleneck to R_1 and maintains its sending rate within the neighborhood of the average fair rate $400/52 = 7.7$ mb/s. At time 80 seconds, 500 short flows start arriving at R_2. This compels the long flow to change its bottleneck to R_2 and converge to the new fair rate. Finally, after all mice flows terminate, the long flow re-stabilizes its sending rate at the capacity of R_2. It can also be seen from Fig. 38(b) that the queue length of link R_1 is kept very small.

7.3 Effect of Random Packet Drops

In this subsection, we examine the performance of JetMax in lossy environments (e.g., wireless networks) with random non-congestion-related packet drops. We first note

that JetMax is not sensitive to packet loss in the return path since out of the ACKs generated in the same Δ_l interval, only one is utilized by the end-user to adjust its sending rate and all others are ignored since they carry duplicate information. We verified this in `ns2` simulations, where the performance of JetMax in $\mathcal{T}_1$ with 90% packet loss in its return path was almost identical to that in the loss-free environment previously shown in Fig. 35(a). We omit the plot of this simulation for brevity and focus on more interesting cases of forward-path loss.

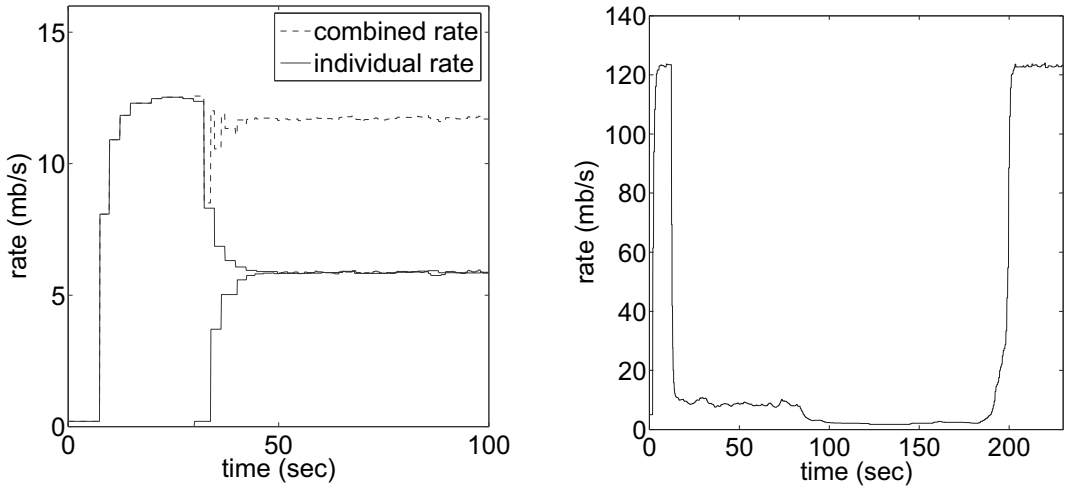
To better see the effect of random loss in the forward path, consider the `ns2` simulation illustrated in Fig. 39(a), where we use $\mathcal{T}_1$ and create 10% and 20% packet loss in the forward paths of flows x_1 and x_2, respectively. As shown in the figure, both fairness and stability are not affected by the forward-path random loss; however, the stationary rates are. To explain this phenomenon, assume $1 - \alpha_{r,l}$ is the total (long-term average) packet loss suffered by flow r along its path to router l. Using Lemma 19, it is not difficult to obtain that

$$x_r^* = \frac{\gamma_l C_l - u_l^*}{\alpha_l N_l}, \tag{151}$$

where α_l is given by

$$\alpha_l = \frac{\sum_{r \in S_l} \alpha_{r,l}}{N_l}, \tag{152}$$

and S_l is the set of responsive flows with respect to link l. Accordingly, we have that the stationary rate x_1^* before the second flow joins the network is $10/0.8 = 12.5$ mb/s, while afterwards both x_1^* and x_2^* are $5/0.85 = 5.82$ mb/s, all of which matches simulation results perfectly. Since only fraction $\alpha_{r,l}$ of flow r's packets survive before arriving into link l, the actual input rate $x_{r,l}^*$ of flow r at l is $x_{r,l}^* = \alpha_{r,l} x_r^*$. This, combined with (151)-(152), leads to $y_l^* = \gamma_l C_l - u_l^*$. Simply put, although the combined sending rate perceived by the end-users may exceed the link's capacity, the



(a) (b)

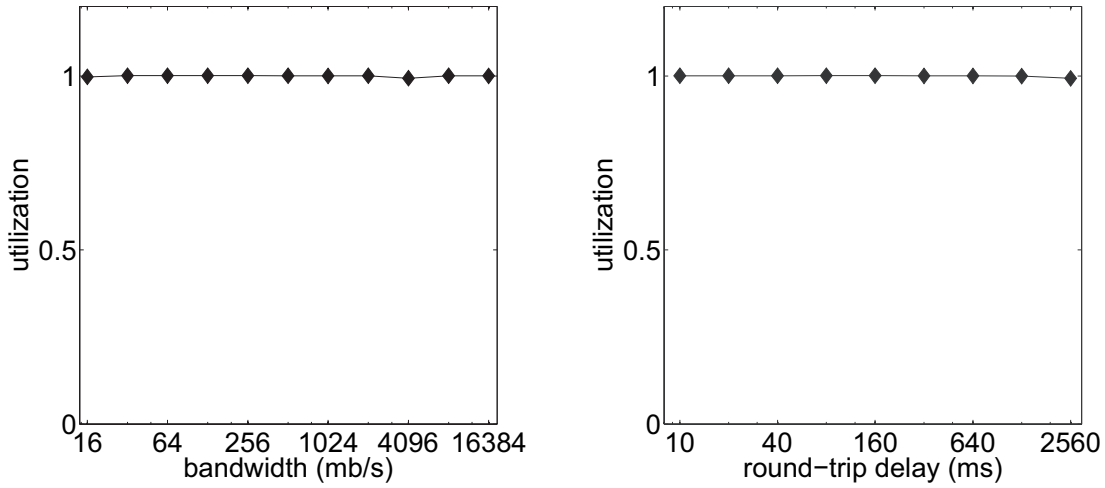
Fig. 39. JetMax ($\tau = 0.6$ and $\gamma = 1$) under random packet loss: (a) $\mathcal{T}_1$ with 10% forward-path loss; (b) "parking lot" topology with mice flows and random loss.

bottleneck link is ideally utilized and free from congestion-related packet loss.

In the next simulation, we test JetMax in the "parking lot" topology used in Fig. 38(b) with 500 mice flows per link, 10% random loss on each link in the forward path, and 50% loss in the backward path. Fig. 39(b) shows the dynamics of the long flow and confirms that JetMax is stable and convergent to the equilibrium as expected.

7.4 Utilization

We next study the effect of link capacity C and round-trip delay on bottleneck utilization of JetMax. We use a single-link topology with 50 flows in the forward direction and 50 flows in the reverse direction. Round-trip propagation delays of these 100 flows are uniformly distributed between $[20, 220]$ ms. We set the target utilization level γ to 1. We also neglect the first 20 seconds of the simulations to avoid transient phase effects and bottleneck switching, and compute efficiency statistics by averaging



(a) link capacity (b) round-trip delay

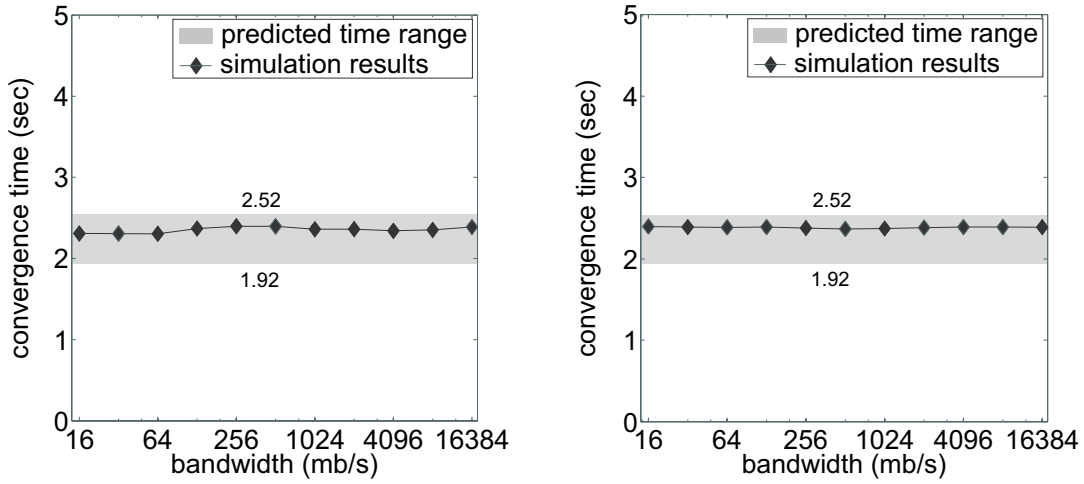
Fig. 40. Utilization of the bottleneck in JetMax ($\tau = 0.6$ and $\gamma = 1$) under different link capacities and round-trip delays in `ns2`.

the instantaneous link utilization sampled every 100 ms in the router.

First, we fix the bottleneck link delay to be 20 ms and vary its capacity from 16 mb/s to 16 gb/s. As Fig. 40(a) shows, JetMax achieves ideal utilization and never overshoots C. Next, we fix the bottleneck capacity to be 1024 mb/s and vary round-trip delays between 10 ms and 2560 ms. From Fig. 40(b), one can observe that JetMax is able to sustain high utilization that does not depend on the RTT. We also note that variance of the instantaneous bottleneck utilization is less than 10^{-3} in all simulations presented in this subsection.

7.5 Convergence Speed

In this subsection, we measure the convergence time of JetMax to $(1-\varepsilon)$-efficiency and $(1-\varepsilon)$-fairness in a single-link topology. We set control gain τ to 0.6, round-trip delay D of all flows to 220 ms, and control interval Δ_l of the bottleneck router to 100 ms. Notice that as discussed in Section 6.5, proper calculation of the reference



(a) 99%-efficiency (b) 99%-fairness

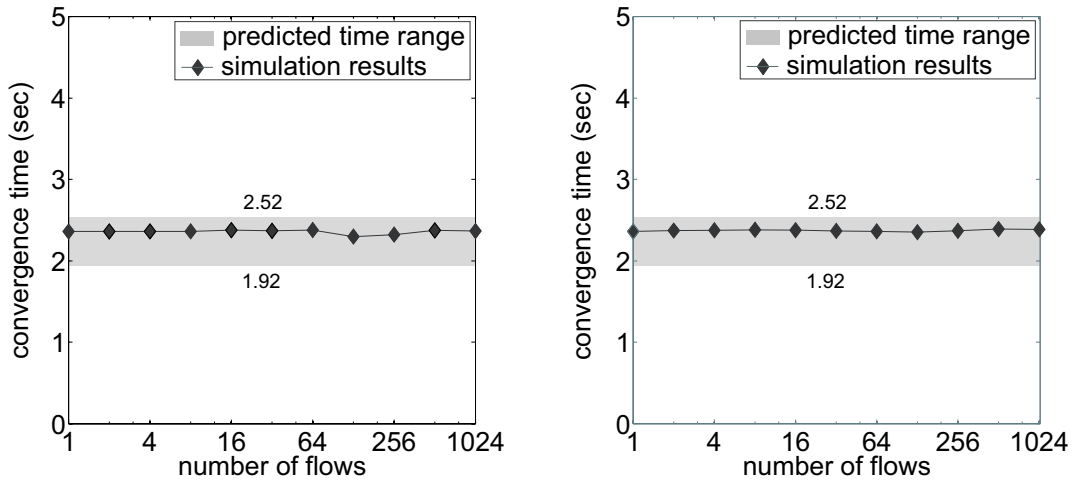
Fig. 41. Convergence time of JetMax ($\tau = 0.6$ and $\gamma = 1$) as a function of bottleneck capacity C in `ns2`.

rate takes $RTT + (1 + k)\Delta_l$ time units, where $k \in [0, 1]$. According to Theorem 17, JetMax converges to 99%-efficiency and 99%-fairness both in $\lceil \log_{|1-\tau|} \varepsilon \rceil = 6$ steps, which corresponds to a time range of $[1.92, 2.52]$ seconds. This prediction is confirmed by simulation results illustrated in Fig. 41, where end-users spend around 2.4 seconds before reaching both efficiency and fairness over a wide range of link bandwidths.

Additionally, Theorem 17 indicates that the convergence rate of JetMax is independent of the number of flows in the system. We also examine this result via `ns2` simulations and verify that, as illustrated in Fig. 42, under different numbers of flows (from 1 to 1024), it takes JetMax the same 6 steps to enter the 1%-neighborhood of both efficiency and fairness.

8 Linux Performance

We finish the chapter by examining performance and implementation overhead of JetMax in Linux software routers. The main goal of this study is to advance beyond

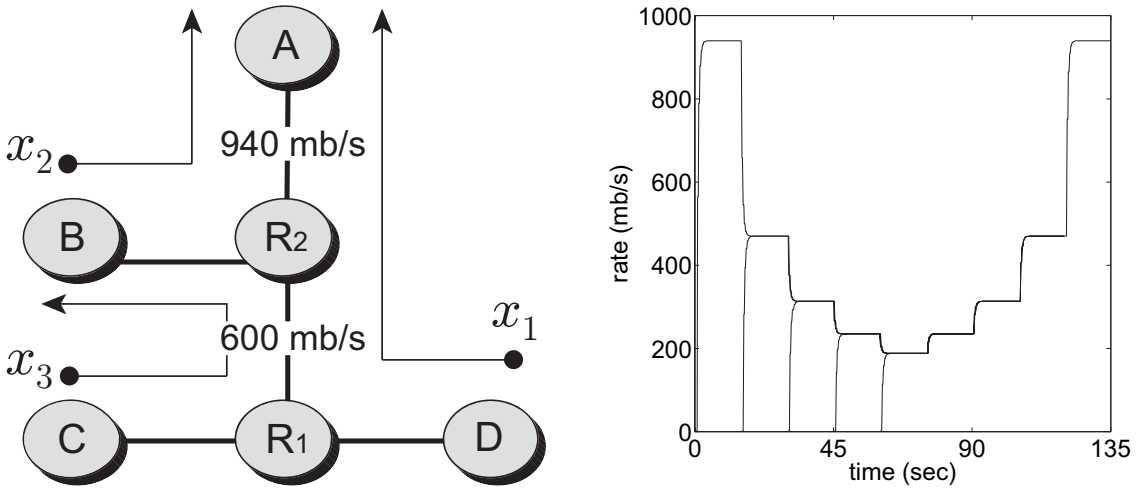


(a) 99%-efficiency (b) 99%-fairness

Fig. 42. Convergence time of JetMax ($\tau = 0.6$ and $\gamma = 1$) as a function of the number of users n in `ns2`.

10 mb/s cases studied in the literature [114] and achieve true gigabit speeds where AQM algorithms would have the most impact in practice. For the experiments reported in this chapter, we use two Linux routers shown in Fig. 43(a), where R_1 is a single Pentium 4 running at 3.4 GHz and R_2 is a dual-Xeon box running at 3 GHz. Propagation delays of links $R_1 - R_2$ and $R_2 - A$ are both 10 ms. Transmit and receive queue lengths of R_1 and R_2 are both set to 2000 packets. All network cards are 1 gb/s full-duplex 1000BaseT Ethernet utilizing PCI-X slots in the their respective computers. Network capacity in the figure is in terms of *transport-layer* rates and is configured independently for each link at 600 and 940 mb/s using different target utilization levels γ_l.

We implemented JetMax in Linux 2.6.9 and built a separately loadable JetMax module that was invoked by netfilter hooks upon each packet queuing event. This module was a standalone application that could be compiled, loaded, and unloaded without rebooting the system. During our investigation, we found that recent

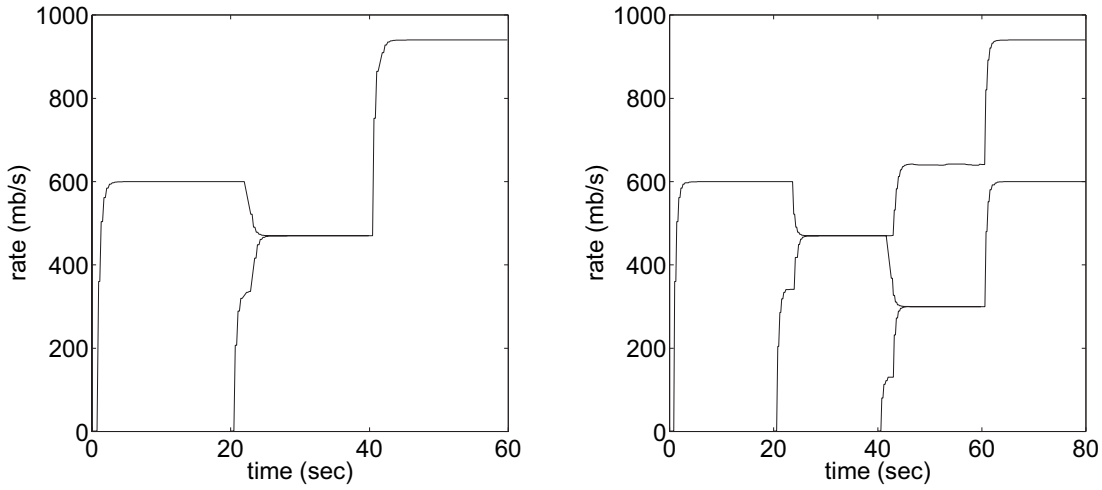


(a) topology (b) single-router test

Fig. 43. Single-router Linux experiments with JetMax ($\tau = 0.6$).

Linux kernels do in fact support floating-point operations (despite a popular belief to the contrary [114]) and that kernel timers are scheduled with remarkable accuracy (i.e., 100 μs), both of which provide significant benefit to AQM algorithms as they often require computation of feedback with high precision and accurate Δ-interval timing.

For the first test, we run five flows from host B to A in Fig. 43(a) to examine the ability of JetMax to utilize high-bandwidth links and support multiple senders/receivers per end-host. Each flow starts with a 15-second delay and lasts for 75 seconds. The performance of JetMax for this setup is shown in Fig. 43(b). Notice in the figure that the first flow converges to 99% of 940 mb/s in 1.3 seconds and maintains its steady-state rate without oscillations. As subsequent flows arrive, they take 1.2 seconds (which is 6 control steps of $\Delta = 200$ ms units each) to achieve 0.99-fairness, where transitions between the neighboring states take place monotonically and the system's combined rate never exceeds 940 mb/s. Similar performance is observed when flows depart, where the system takes approximately 1.2 seconds to



(a) two-flow test (b) three-flow test

Fig. 44. Dual-router Linux experiments with JetMax ($\tau = 0.6$).

re-stabilize each time.

We next test JetMax's capability of managing bottleneck switching in multi-link scenarios. We start flows x_1 and x_2 in Fig. 43(a) with a 20-second delay. Notice that x_1 should first converge to 600 mb/s, then shift its bottleneck to R_2, and eventually settle down at 470 mb/s. This is shown in Fig. 44(a), where the flows perform precisely as expected. When flow x_1 departs at $t = 40$, x_2 quickly converges to 940 mb/s.

In our final setup, we repeat the same experiment except that flow x_3 joins at time $t = 40$ seconds. This allows the bottleneck of flow x_1 to shift twice during its stay in the system. The corresponding simulation result is illustrated in Fig. 44(b), where x_1 and x_2 first converge to 470 mb/s each and maintain this rate until $t = 40$. When x_3 joins, it quickly settles down with x_1 at 300 mb/s and x_2 takes the remaining bandwidth (i.e., 640 mb/s). Once x_1 departs at $t = 60$, x_2 converges to 940 mb/s and x_3 to 600 mb/s. Notice that in this experiment router R_2 delivers over 1.5 gb/s combined throughput to end-flows without losing any packets.

9 Discussion

This chapter examined several max-min AQM congestion controllers and found that all of them exhibited undesirable properties under certain criteria. A bigger problem, however, discovered in this work was the susceptibility of XCP and potentially other max-min systems with non-monotonic feedback to oscillation between bottlenecks and unstable behavior in multi-router topologies. We proposed a new method JetMax that was able to overcome the identified issues with existing methods and admitted multi-link stability (to the extent examined in this study), fast convergence to efficiency/fairness, loss-free dynamics, adjustable link utilization, and simple implementation. We note that multi-link stability analysis conducted in this chapter is limited in scenarios where bottleneck assignments are consistent and time-invariant. We leave a rigorous study of the bottleneck-switching problem in generic max-min methods for the future. In addition, performing a more extensive experimental evaluation of JetMax and designing its simplifications form other lines of our planned work.

CHAPTER VII

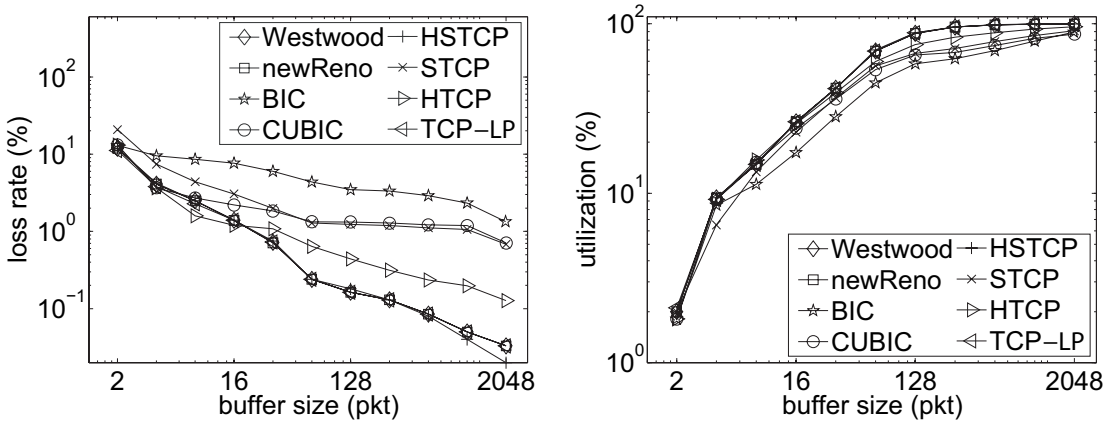
ADAPTIVE BUFFER SIZING (ABS)

1 Motivation

While a comprehensive modeling of Internet traffic and its relationship with buffer size b remains open, we show in this subsection that there are strong indications that there exists a *monotonic* relationship between b and two key performance metrics, loss rate p and utilization u. Due to the extreme difficulty of the problem, we do not seek to present a rigorous proof of this monotonic relationship, but provide an intuitive explanation experimentally via `ns2` simulations and analytically using a simple congestion control model. This monotonic relationship serves as motivation and foundation of our adaptive buffer sizing scheme proposed in the following section.

1.1 Simulation Illustration

We next empirically examine the impact of the buffer size on the performance of different congestion control protocols using `ns2` simulations. To accomplish this goal, we utilize the framework developed in [101], which incorporates into `ns2` the Linux-2.6.16.3 implementations of several proposed TCP variants, including newReno [32], BIC [110], CUBIC [89], HSTCP [30], HTCP [69], STCP [61], Westwood [38], and TCP-LP [67]. The simulation setup is composed of one bottleneck link of capacity 100 mb/s and ten sources with packet size 1500 bytes and RTTs uniformly distributed in $[30, 50]$ ms. We set buffers of access links to be 2500 packets and verify that no packet is lost at these links in all simulations. The plots of loss rate p and utilization u under different bottleneck buffer sizes are given in Figure 45. As expected, both p



(a) loss rate (b) utilization

Fig. 45. Effect of buffer size b on loss rate p and utilization u of several TCP variants.

and u are monotonic functions of b. Note that the loss curves of CUBIC and STCP in Figure 45(a) appear to be flat when b is between 64 and 1024 packets; however, we verified numerically that they are actually monotonically decreasing.

1.2 Intuition

This monotonic relationship is not surprising. It is intuitive clear that a larger buffer can absorb more bursts in packet arrivals, thereby reducing the frequency of packet drops. In addition, assuming the buffer is always depleted between events of packet drops, it takes longer time for sources to saturate a larger buffer so that to experience the next packet loss. Thus, the average loss rate is a decreasing function of the buffer size. At the same time, a larger buffer can hold more packets and thus maintain the bottleneck link at full utilization for a longer time when senders back off in response to congestion. Therefore, the average utilization level proportionally scales with the buffer size. However, the above reasonings assume a depleted queue after each packet loss and may not be obvious otherwise. Thus, it is desirable if we can obtain a more generic explanation with less restrictive assumptions.

1.3 Exogenous Traffic

In this subsection, we consider two popular traffic models, Poisson and self-similar packet arrivals. It has been shown in [17] that with increase in the link capacity and number of flows accessing the router, packet arrivals tend to be a Poisson process, i.e., packet inter-arrival time is exponentially distributed. Thus, assuming packet sizes are also exponentially distributed, the queuing process can be modeled by an $M/M/1/b$ queue. Then, loss probability is expressed by $p = (1-u)u^b/(1-u^{b+1})$, where $u = \lambda/C$ [90]. Noticing that $1 - u^{b+1} = (1 - u)\sum_{i=0}^{b} u^i$, we obtain $p = 1/\sum_{i=0}^{b} u^{-i}$, which, combined with the fact that u is a constant, immediately implies that p is monotonically decreasing in b.

The relationship between buffer size b and loss probability p under self-similar input traffic has been studied by Gong *et al.* [40] in ATM networks. Specifically, consider a single router with buffer size b and N independent traffic sources. The arrival process of each traffic source is modeled as an *Interrupted Bernoulli Process*. Then, the system can be modeled as a discrete-time Markov chain with state $[X_i, Y_i]$, where, at time i, X_i denotes the number of cells in the buffer and Y_i denotes the number of active sources. Then, cell loss probability can be calculated as follows:

$$p = \frac{\sum_{(m,n)\in S}(m + n - b)^+ Pr\{X = m, Y = n\}}{\sum_{(m,n)\in S} mPr\{X = m, Y = n\}}, \tag{153}$$

where S is the state space of the Markov chain and $(x)^+ = \max(x, 0)$. From the last equation, it is evident that cell loss rate p monotonically decreases as buffer size b increases. A similar result under a different self-similar traffic model is later obtained by Likhanov *et al.* [70].

1.4 Endogenous Traffic

We note that the goal of this section is not to present a comprehensive congestion control model that is able to formulate generic Internet traffic, but to intuitively explain results observed in the previous section using a simple, but illustrative, model. We start with the definition of p and u in mathematical terms. Consider a scenario where traffic passes through a single-channel FIFO queue of capacity b and service rate C. Let $L(t)$ and $A(t)$ respectively denote the number of lost and admitted packets by time t. Then, p is defined as the *long-term* average loss rate $p = \lim_{t\to\infty} L(t)/(L(t) + A(t))$, $u = \lim_{t\to\infty} A(t)/Ct$ as the average utilization, and $\lambda = \lim_{t\to\infty} A(t)/t$ as the average input rate.

Using these definitions, we next examine the effect of buffer size on traffic that reacts to congestion using a simple model. Denoting by $W_i(n)$ the congestion window size of flow i during the n-th RTT, we can model a generic congestion control algorithm as following[1]:

$$W_i(n) = \begin{cases} W_i(n-1) + \alpha_i\big(W_i(n-1)\big) & \text{no loss} \\ W_i(n-1) - \beta_i\big(W_i(n-1)\big) & \text{loss} \end{cases}, \tag{154}$$

where $\alpha_i(\cdot)$ and $\beta_i(\cdot)$ are non-negative functions and each discrete time step corresponds to one RTT. We note that this model is capable of describing the start-up and congestion-avoidance behavior of responsive flows and keep in mind that traffic that is unresponsive to congestion can be modeled as exogenous as discussed in the preceding subsection.

In particular, framework (154) subsumes a wide spectrum of loss-based congestion control protocols, including AIMD (e.g., Reno [1] and Westwood [38]), MIMD

[1]We assume $W_i(n)$ are rounded to integers during calculations and omit the corresponding ceiling function for brevity.

(e.g., Scalable TCP [61]), and many other existing TCP variants (e.g., BIC [110], TCP-LP [67], and HSTCP [30]). Note that delay-based schemes (e.g., FAST [49] and Vegas [15]) generally are not sensitive to buffer size b as long as it is kept larger than the stationary queue length q^* of the system. However, when $b < q^*$ these methods experience packet losses and their responses can also be modeled by (154). Moreover, since (154) allows different response functions $\alpha_i(\cdot)$ and $\beta_i(\cdot)$ for different flows i, this model applies to scenarios where the traffic is generated by a *mixture* of protocols.

Assuming N sources with homogeneous RTTs access a single link of capacity C and letting $q(n)$ be the queue length at time n, we can model the queuing dynamics using recurrence:

$$q(n) = \min\big((q(n-1) + X(n) - C)^+, b\big), \tag{155}$$

where b is the buffer size and $X(n) = \sum_{i=1}^{N} W_i(n)$ is the total number of arrivals during the n-th RTT. Assuming that $W_i(n)$ of each source i is bounded above by W_{max}, the system dynamics can be represented by a discrete Markov chain with state $S_n = [W_1(n), W_2(n), \ldots, W_N(n)]$ and state space $O : [1, 2, \ldots, W_{max}]^N$.

Let $Z(n)$ be the number of dropped packets during the n-th RTT: $Z(n) = \big(q(n-1) + X(n) - C - b\big)^+$. Define $v(n) = Z(n)/X(n)$ as the average loss rate during this RTT. Assuming packet loss rate of each flow is independent of each other, we have $Pr\{W_i(n+1) = W_i(n) + \alpha_i(W_i(n))\} = (1 - v(n))^{W_i(n)}$ and $Pr\{W_i(n+1) = W_i(n) - \beta_i(W_i(n))\} = 1 - (1 - v(n))^{W_i(n)}$. Based on this, we can derive the transition probability between any pair of states. Furthermore, it is clear that the transition probability depends only on the previous state, which implies that series $\{S_n\}$ is a Markov chain. Then, the following result is easy to obtain.

Theorem 18. *The Markov chain defined by* (154)-(155) *always converges to a stationarity distribution.*

According to Theorem 18, for any fixed N and starting from any initial state, the Markov chain defined by (154)-(155) always converges to its steady state. Thus, we omit the transient phase in the rest of the section and only examine the queuing process under traffic generated by a stationary Markov chain. In such a scenario, the following result shows that packet loss rate p scales inversely proportionally to the buffer size b.

Theorem 19. *Loss probability p in a finite queue fed by traffic governed by a stationary Markov chain defined by* (154)-(155) *monotonically decreases in queue size b.*

Proof. Under the stationary Markov chain defined by (154)-(155), denote by S_n the state at time n, by M the number of states, and by $\boldsymbol{\pi} = [\pi_1, \pi_2, \ldots, \pi_M]$ the stationary probability vector of each state (i.e., $\pi_i = Pr\{S_n = i\}$). Further let $A_{i,j}(k)$ be the probability that the chain goes from state i to j and the next arrival has k packets, i.e., $A_{i,j}(k) = Pr\{X(n+1) = k, S_{n+1} = j | S_n = i\}$. Then, define $\boldsymbol{A}_k$ as the probability matrix whose (i,j)-th element is $A_{i,j}(k)$. Using these variables, we can represent the traffic density ρ as $\rho = \boldsymbol{\pi} \sum_{k=1}^{\infty} k\boldsymbol{A}_k \boldsymbol{e}$, where $\boldsymbol{e}$ is a column vector with all elements equal to one.

We next express loss probability of a finite buffer of size b in terms of the queue length distribution of an infinite buffer, whose queuing process $q'(n)$ is given by:

$$q'(n) = \left(q'(n-1) + X(n-1) - c\right)^+. \tag{156}$$

Consider the steady-state probability matrix $\boldsymbol{\Delta}$ of the infinite buffer, where the (k,i)-th element $\Delta_{k,i}$ is:

$$\Delta_{k,i} = \lim_{n\to\infty} Pr\{X(n) = k, S_n = i\}. \tag{157}$$

Then, according to [46, Theorem 4], under arrivals governed by the same stationary

Markov chain, loss probability p of a finite buffer of size b is represented by:

$$p = \frac{(1-\rho)\sum_{k=b+1}^{\infty} \boldsymbol{\Delta}_k \boldsymbol{A}_0 \boldsymbol{e}}{\rho \sum_{k=0}^{b} \boldsymbol{\Delta}_k \boldsymbol{A}_0 \boldsymbol{e}}, \tag{158}$$

where $\boldsymbol{\Delta}_k$ is the k-th row vector of $\boldsymbol{\Delta}$. Since the Markov chain is stationary, all variables in the last equation are constant. Thus, it follows that loss rate p monotonically decreases as buffer size b increases. □

As an example of Theorem 19, we examine the relationship between b and p of TCP. Analytical modeling of TCP has been an active research topic since the last decade [13, 78, 81, 105]. Specifically, it is derived in [105] that loss probability p is given as follows:

$$p = \frac{8N^2}{3(CR+b)^2}. \tag{159}$$

A similar result is obtained in [13] for geostationary satellite networks:

$$p = \frac{128N^2}{27(CR+b)^2}. \tag{160}$$

Both results indicate that p monotonically decreases as b grows. We also note that Theorem 19 assumes homogeneous RTTs. As verified later in simulations, the same result holds for heterogeneous RTTs.

Moreover, it is clear that utilization u scales inversely to loss rate p according to a general TCP model of the form $r = c/p^d$, where r is the throughput and c and d are constants [110]. This holds for various TCP flavors including Reno, BIC, HSTCP, and STCP. Further notice that according to Theorem 19, loss rate p scales inversely to b. This implies that utilization u monotonically increases in b. Thus, in the rest of this chapter we assume the monotonic relationship between u and b.

In addition, it is worth noting that although system (154) is generic enough to represent the increase/decrease behavior of a wide class of congestion control al-

gorithms, it is by no means comprehensive. To make the model more complete, one should additionally consider factors such as slow-start, timeouts, and retransmissions. To maintain a proper scope of the chapter, we leave further investigation of this problem for future work and next demonstrate via `ns2` simulations that the obtained properties indeed exist in various current congestion control protocols.

2 Adaptive Buffer Sizing (ABS)

In this section, we describe a dynamic buffer sizing framework that is adaptive to dynamics and uncertainties of input traffic while maintaining the system under target performance constraints such as loss rate p^* and utilization u^*. As an example of this framework, we start with a simple mechanism, progressively identify and overcome its underlying drawbacks, and eventually arrive at the final controller that we call ABS.

2.1 General Consideration

To design a buffer sizing mechanism, first it is necessary to understand how buffers are managed in current commercial routers. The memory system in a Cisco 3600 series router [91], for instance, is composed of the main processor memory, shared (packet) memory, flash memory, nonvolatile random access memory (NVRAM), and erasable programmable read only memory (EPROM). Among them, we are particularly interested in shared (packet) memory, which is used for packet buffering by the router's network interfaces. Specifically, each interface is associated with an input hold buffer (IHB), which resides in the system buffer of shared memory and is used to store packets for transfer between fast switching and process switching code. For each packet arriving into an interface, the interface driver writes it into an IHB. An

incoming packet is immediately dropped if the IHB reaches its maximum size, which is static and does not grow or shrink based on demands. Our goal in this section is redesign IHB such that its size adapts to dynamics of the incoming traffic.

We note that it is important to distinguish the framework of dynamic buffer sizing from the large class of AQM algorithms (e.g., RED [33], REM [4], and PI [44]). These methods operate within a given buffer size b_l and aim to stabilize the queue occupancy (or queuing delay) at a certain target level, which is a portion of the selected buffer size b_l. Thus, AQM is unable to solve issues associated with incorrectly sized router buffers.

To better see this, we test several TCP variants under REM using `ns2` simulations. Recall that an REM-enabled router dynamically updates its packet dropping/marking probability by monitoring the discrepancy between the aggregate input rate $y(t)$ and link capacity C and the difference between the current queue length $q(t)$ and its target value q^*. In the steady state, the system achieves $y(t) = C$ and $q(t) = q^*$. In our simulations, we use a simple "dumbbell" topology with a single REM ($q^* = 50$ pkts) link of capacity $C = 10$ ms/b shared by 20 TCP sessions. We use marking at the REM router and enable ECN at end-users. As seen from Table I, if we set buffer size of the bottleneck link to $b = 100$ pkts, which is greater than REM's target queue size $q^* = 50$ pkts, REM successfully maintain the queue size close to q^* while achieving 100% utilization and 0% packet loss for all TCP variants. However, if we set buffer size b below q^*, the bottleneck link suffers significant under-utilization and prohibitively high loss rate.

In contrast, dynamic buffer-sizing mechanisms focus on determining the optimal size of the physical buffer. This way, the router can efficiently allocate its available buffers among different memory-sharing interfaces, hereby achieving pre-agreed QoS requirements, shrinking the required space of the main memory, and reducing the

Table I. Performance of different TCP variants with REM ($q^* = 50$ pkts) under different buffer sizes.

	$b = 100$ pkts			$b = 10$ pkts		
	q (pkts)	p (%)	u (%)	q (pkts)	p (%)	u (%)
Reno	56.14	0.00	100.00	5.41	9.88	84.78
BIC	52.88	0.00	100.00	5.27	9.04	86.52
CUBIC	52.48	0.00	100.00	4.92	7.84	87.22
HSTCP	56.73	0.00	100.00	4.96	9.60	86.59
STCP	54.38	0.00	100.00	5.40	12.48	83.74
HTCP	57.77	0.00	100.00	4.60	8.20	87.46
Westwood	54.61	0.00	100.00	4.99	10.08	84.50

system cost and board space. Dynamic buffer sizing schemes can overcome the problem of improper buffer sizing that AQM is unable to solve or may be combined with AQM methods to achieve desired performance. In existing Internet routers where memory is already fixed, the proposed approach is also valuable since it guarantees the minimum queuing delay in each interface under predetermined performance constraints. Additionally, ABS lends ISPs and router manufactures great freedom in choosing preferred constraints when configuring their routers.

2.2 Controller Design

Although the underlying differential/difference equations describing the effect of router buffer size on Internet traffic remain unknown, it follows from the last section an important property of this relationship – *monotonicity*. This implies that for any given feasible loss rate p^* (or utilization u^*) under stationary input traffic, there exists a

unique buffer size b^* such that the resulting system achieves p^* (or u^*). In addition, this monotonic relationship gives us a hint of the correct direction in which we should adjust the buffer size. Specifically, assuming target loss rate p^* and its actual value $p(n)$ measured at time n, the router buffer size $b(n)$ should increase if $p(n) > p^*$ and decrease otherwise. Analogously, given u^* and $u(n)$, $b(n)$ should decrease if $u(n) > u^*$ and increase otherwise. This result allows us to develop simple yet robust controllers to adaptively size router buffers to satisfy given system constraints.

One natural candidate for achieving this goal is the Integral controller. First, consider the controller under the loss rate constraint, in which case $b_p(n)$ denotes the buffer size at time n and $e_p(n) = p(n) - p^*$. Then, the Integral controller can be represented by its z-domain transfer function $G_p(z)$:

$$G_p(z) = \frac{B_p(z)}{E_p(z)} = \frac{I_p T}{1 - z^{-1}}, \tag{161}$$

where $B_p(z)$ and $E_p(z)$ are respectively z-transforms of $b_p(n)$ and $e_p(n)$, integral gain I_p is a positive constant, and T is the sampling interval. Rewriting (161) in the time domain, we arrive at the following difference equation:

$$b_p(n) = b_p(n-1) + I_p T\big(p(n) - p^*\big), \tag{162}$$

where T is the sampling interval and I_p is the integral gain. Similarly, we obtain the control equation of $b_u(n)$ under the utilization constraint:[2]

$$b_u(n) = b_u(n-1) - I_u T\big(u(n) - u^*\big), \tag{163}$$

where I_u is the integral gain. It is noteworthy to point out that since $p(n)$ monoton-

[2]Additional constraints, such as queuing delay, can be easily adopted in our method. For ease of presentation, we only concentrate on loss rate and link utilization in this chapter.

ically decreases with $b(n)$ while $u(n)$ increases with $b(n)$, controllers (162) and (164) have opposite signs before their respective integral gain.

However, the last controller invites a serious problem if deployed in non-bottleneck routers. This is because a non-bottleneck router is always under-utilized regardless of its buffer size. Thus, if u^* is set to be above the maximally achievable utilization level of the link, the router always have $u(n) < u^*$ and drives its buffer size to infinity. We overcome this problem by modifying (163) as follows:

$$b_u(n) = b_u(n-1) - I_u T\big(u(n) - u(n-1)\big)\big(u(n) - u^*\big). \tag{164}$$

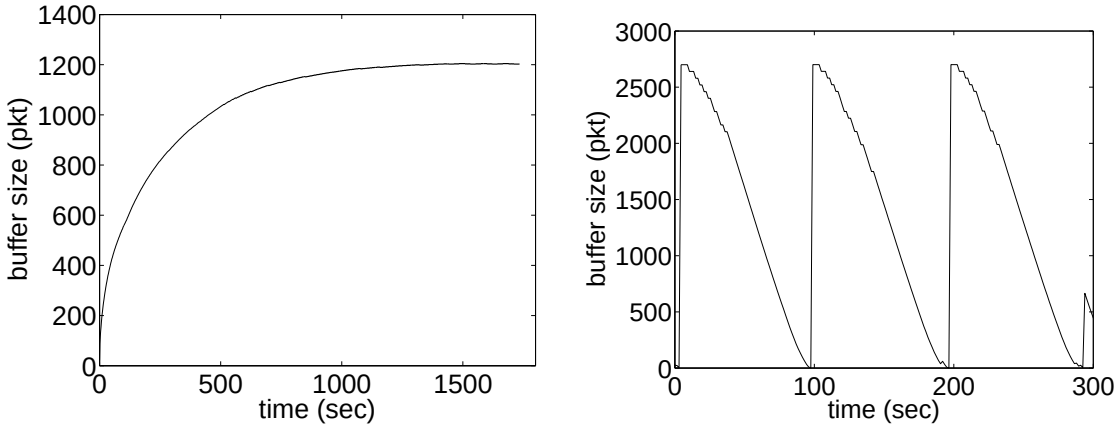
Compared to (163), the extra term $u(n) - u(n-1)$ in (164) is to damp the effect of $(u(n) - u^*)$. Specifically, in the steady state of a non-bottleneck router, we must have $u(n) - u(n-1) = 0$, which forces the second term of (164) to converge to zero and prevents $b_u(n)$ from diverging to infinity.

Now, we have two buffer sizes $b_u(n)$ and $b_p(n)$ based on the utilization and loss rate constraints, respectively. Similar to BSCL [25] , the minimum buffer size $b(n)$ satisfying both requirements should be the larger of $b_p(n)$ and $b_u(n)$, i.e.,

$$b(n) = \max\big(b_u(n), b_p(n)\big). \tag{165}$$

We call the hybrid controller (162)-(165) Adaptive Buffer Sizing (ABS) scheme and sub-controllers (162) and (164) ABS_p and ABS_u, respectively. In practice, these two controllers can be used together or separately. Note that ABS does not rely on comprehensive prior knowledge of the system being controlled, but adapts the controller according to errors of the output signals $u(n)$ and $p(n)$. As a consequence, ABS is expected to work in practical network settings and be robust to real Internet traffic (more on this below).

This controller works very well in many cases. However, its main limitation lies



(a) $u^* = 95\%$, $p^* = 0.5\%$ (b) $u^* = 70\%$, $p^* = 5\%$

Fig. 46. ABS ($I_u = I_p = 3000$ and $T = 200$ ms) without parameter training in a network with a single link of capacity 10 mb/s and 20 TCP flows.

in the difficulty in choosing the optimal gain parameters I_u and I_p. Specifically, if they are chosen too small, the system may suffer from a sluggish convergence rate to the equilibrium; however, if they are set too large, the system may exhibit exceedingly aggressive adaptation behavior and persistently oscillate around, instead of converging to, the stationary point. This phenomenon is illustrated in Figure 46, where 20 TCP flows share an ABS-equipped bottleneck link of capacity 10 mb/s. We set integral gains $I_p = I_u = 3000$ and control interval $T = 200$ ms. As seen in the figures, ABS is stable and converges the buffer size to 1200 packets when $u^* = 95\%$ and $p^* = 0.5\%$. However, when $u^* = 70\%$ and $p^* = 5\%$ the system becomes unstable and the buffer size periodically oscillates between 1 and 2700 packets.

Due to the lack of the knowledge of the differential equations describing the system, it is unlikely that any off-line pre-training of the controller's parameters I_u and I_p can be effective. Even if such a method could exist, parameters trained for a particular system setting may immediately become inappropriate as the traffic dynamics evolve. We next seek to overcome this issue by designing a parameters

tuning mechanism that is able to adaptively converge the control parameters to their optimal values for the current system configuration.

2.3 Adaptive Parameters Training

It is clearly a non-trivial task to find the *optimal* parameters for controlling such a complex system as the Internet, which is especially the case provided that the system has an *unknown* underlying model and dynamically changes over time. However, we manage to achieve this goal using a simple scheme, which combines the *output error* [2] method and the *gradient descent* [93] technique. In what follows, we explain our method in the context of ABS_u and note that the mechanism for ABS_p can be obtained similarly.

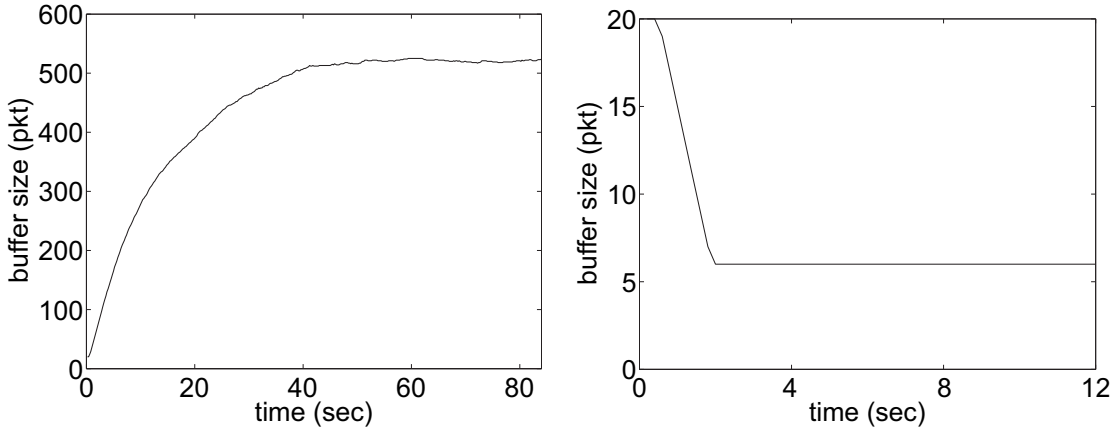
Denote by $I_u(n)$ the instantaneous value of integral gain I_u at time n. Then, rewrite ABS_u's control equation (164) as:

$$b_u(n) = f_u\big(u^*, b_u(n-1), u(n), I_u(n)\big), \tag{166}$$

where function $f_u(.)$ is given by the right-hand side of (164). Suppose that the router, at the end of the n-th control interval, sets its buffer size to $b_u(n)$ based on (166) and observes that link utilization becomes $u(n+1)$ during the next interval. Then, we know that if we set $u^* = u(n+1)$ as the target utilization, $b_u(n)$ must be the *optimal* output of controller (166) under the same traffic pattern and given buffer size $b_u(n-1)$ and utilization $u(n)$. This is equivalent to saying that under the optimal control gain I_u^*, we must have the following equation:

$$b_u(n) = f_u\big(u(n+1), b_u(n-1), u(n), I_u^*\big). \tag{167}$$

Thus, at every control step, we get the exact value of the inverse function of the controlled plant [2]. Hence, it remains to adaptively adjust $I_u(n)$ to achieve its



(a) $u^* = 95\%$, $p^* = 0.5\%$ (b) $u^* = 70\%$, $p^* = 5\%$

Fig. 47. ABS with parameter training in a network with a single link of capacity 10 mb/s and 20 TCP flows.

optimal value I_u^*, which translates into the following optimization problem.

First, define $b'_u(n) = f_u\big(u(n+1), b_u(n-1), u(n), I_u(n)\big)$ as the actual controller's output under the current integral gain $I_u(n)$. This value is not used to decide the buffer size, but is applied to the following calculation: $F_u(n) = b'_u(n) - b_u(n)$, which is the difference between the actual and optimal outputs. Then, the optimal control gain I_u^* under the current traffic is the one that minimizes $F_u(n)$. To find this optimal parameter, we use the gradient-descent algorithm.

According to the gradient-decent method, since function $F_u(n)$ is differentiable at $I_u(n)$, it decreases fastest along the direction of its gradient $\bigtriangledown F_u(I_u(n))$, which is the derivative of $F_u(n)$ with respect to $I_u(n)$. Thus, at every control step, if the router updates parameter $I_u(n)$ as follows:

$$I_u(n+1) = I_u(n) - \gamma \bigtriangledown F_u\big(I_u(n)\big), \tag{168}$$

with step size γ (which is set to 1 in all simulations presented here), then we have that sequence $F_u(I_u(1)) \geq F_u(I_u(2)) \geq \ldots$, which eventually converges to zero. In

this case, $I_u(n)$ reaches I_u^*.

Invoking (164), we simply have:

$$\bigtriangledown F_u(I_u(n)) = \frac{dF_u(n)}{dI_u(n)} = T\big(u(n+1) - u^*\big). \tag{169}$$

Combining the last two equations, we arrive at the following parameter tuning rule for $I_u(n)$:

$$I_u(n+1) = I_u(n) - \gamma T\big(u(n+1) - u^*\big). \tag{170}$$

Following the above techniques, we can derive the following parameter training rule for $I_p(n)$ in ABS_p:

$$I_p(n+1) = I_p(n) - \gamma T\big(p^* - p(n+1)\big). \tag{171}$$

Thus, we have finished the design process of ABS, which now consists of two basic Integral controllers (162)-(164) and two parameter training components (170)-(171). Note that the resulting system is independent of the exact model of the controlled plant and highly adaptive to the plant's changing system dynamics. In addition, the proposed parameter training mechanism is not limited to our particular case, but applicable to other systems with multiple parameters to be optimized.

To examine performance of the resulting controller, we rerun simulations in Figure 46. The simulation results are illustrated in Figure 47, from which we can see that in both cases ABS successfully converges the buffer size to its stationary value and exhibits much faster convergence speed compared to its original version shown in Figure 46. We note that since traffic loads at Internet routers change slowly over time, buffer sizing schemes are able to utilize long sampling intervals to filter out noise in measurements and achieve more accurate approximation of the systems average behavior. Thus, the convergence rate of ABS should not be confused with that of a

congestion control or AQM protocols, whose control actions are usually performed at the time-scale of milliseconds. However, as we demonstrate later, ABS is in fact very responsive and works well under highly bursty Internet traffic. Finally, we observe in both simulations that gain parameters I_u and I_p indeed converge to their respective optimal value.

3 Performance

We next demonstrate via `ns2` simulations performance of ABS under a wide range of flow populations and link capacities, dynamically changing traffic loads, synthetic web traffic, and mixture of TCP and non-TCP flows.

3.1 Implementation

ABS admits a very simple implementation and incurs minimal computational overhead. Specifically, the router maintains two counters S_1 and S_2 to respectively record the amount of data enqueued and dropped by the router during the current sampling interval. For each incoming packet k with size s_k, either S_1 or S_2 is incremented by s_k depending on whether the packet is admitted. Thus, there is only one addition per packet. At the end of the n-th interval, the router computes loss rate using $p(n) = S_1/(S_1 + S_2)$ and utilization using $u(n) = (S_1 + S_2)/(CT)$ where C is the link's capacity. Then, the router calculates the gain parameters based on (170)-(171) and decides its buffer size according to (162)-(165). Since these operations are performed once every control interval (which is set to 20 seconds in our simulations), the incurred overhead is negligible.

In practice, dynamic buffer sizing may encounter some implementation issues. For instance, one such problem is *memory fragmentation*, which occurs when the

router frequently allocates and releases differently sized memory blocks and as a result the memory space contains a lot of small unused pieces. This problem can be mitigated by increasing granularity of memory allocation, i.e., allocating in fixed-size chunks of memory. Sizes of chunks can be 2048 bytes, 4096 bytes, or other values depending on the system. However, for purpose of demonstration, we do not include this mechanism in simulations shown below.

3.2 Scalability

In this subsection, we compare performance of existing buffer sizing mechanisms (i.e., BDP, Stanford model, BSCL, and ABS) under different link capacities C and flow populations N. Note that due to lack of publicly available implementation and unspecified control parameter K, we do not include ADT in this comparison study. We use a "dumbbell" topology with N long-lived TCP flows, whose RTTs are randomly distributed in $[30, 30 + 2N]$ ms. As suggested in [25], we use the harmonic average RTT R_e for the BDP rule. For the Stanford model, we use equation $b = 2R_eC/\sqrt{N}$. In BSCL, we set the loss synchronization factor α to 0.6. In both BSCL and ABS, we set $u^* = 98\%$ and $p^* = 2\%$.

We first fix link capacity $C = 16$ mb/s and vary N between $[2, 1024]$. The simulation results are illustrated in Table II, in which data are averaged over the second half of each simulation to eliminate initial transient effects. As shown in the table, when the number of flows is small, both the BDP and Stanford rules are not very successful in achieving their design goal (i.e., high link utilization). As N becomes large, both methods do achieve high utilization, but in the expense of high loss rates. This is especially evident in the Stanford model, whose loss rate is 13.45% when there are 1024 flows. Capability of controlling loss rate is improved in BSCL; however, it still cannot achieve the target loss rate $p^* = 2\%$ and suffer from low link

Table II. Performance of existing buffer-sizing strategies in a single-link network with different numbers of flows N ($C = 10$ mb/s, $u^* = 98\%$, and $p^* = 2\%$)

N	BDP			Stanford			BSCL			ABS		
	b (pkts)	p (%)	u (%)	b (pkts)	p (%)	u (%)	b (pkts)	p (%)	u (%)	b (pkts)	p (%)	u (%)
2	110	0.08	94.3	155	0.06	95.8	55	0.11	89.6	287	0.03	97.9
4	116	0.18	94.3	116	0.18	94.3	47	0.32	85.5	232	0.09	97.8
8	128	0.42	93.5	91	0.55	88.5	45	0.83	78.6	229	0.23	97.9
16	150	1.05	94.9	75	1.6	87.8	45	1.8	83.4	227	0.74	97.9
32	190	1.76	98.3	68	2.88	92.8	45	3.24	89.8	151	2.04	97.9
64	261	2.76	99.9	66	4.85	95.8	94	4.51	97.6	443	2.00	100.0
128	383	4.06	100.0	68	7.08	97.5	358	4.18	100.0	941	2.05	100.0
256	595	5.11	100.0	75	9.38	98.8	922	4.24	100.0	2006	2.03	100.0
512	965	5.69	100.0	86	11.42	99.8	2106	4.19	100.0	4210	2.00	100.0
1024	1618	6.47	100.0	102	13.45	100.0	4569	3.94	100.0	8613	2.03	100.0

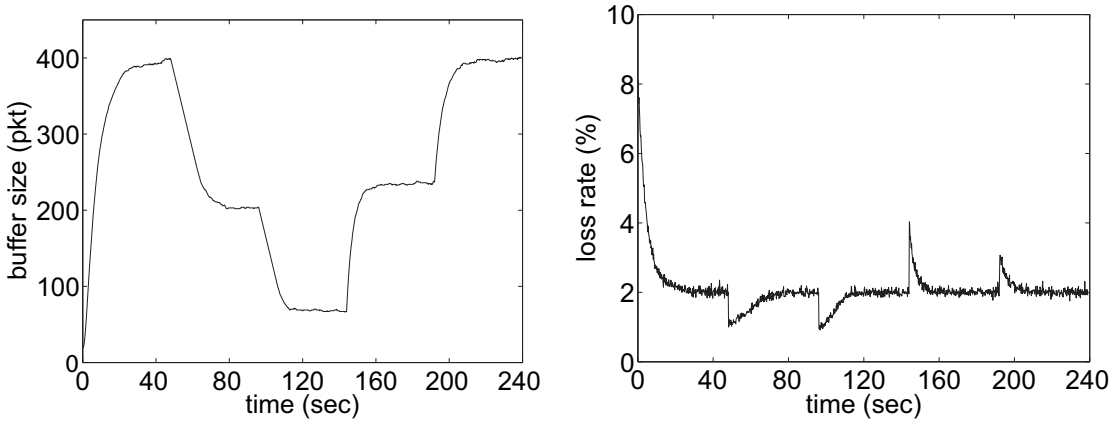
Table III. Performance of existing buffer-sizing strategies in a single-link network with different link capacities C ($N = 16$, $u^* = 98\%$, and $p^* = 2\%$)

C (mb/s)	BDP			Stanford			BSCL			ABS		
	b (pkts)	p (%)	u (%)	b (pkts)	p (%)	u (%)	b (pkts)	p (%)	u (%)	b (pkts)	p (%)	u (%)
2	19	9.7	99.2	10	12.7	97.5	76	3.7	99.9	148	2.08	99.9
4	38	4.8	98.4	19	6.6	95.1	53	4.0	99.1	131	2.03	99.7
8	75	2.4	96.1	38	3.03	91.9	17	4.3	79.5	120	1.85	97.9
16	150	1.05	94.9	75	1.6	87.8	45	1.8	83.4	227	0.74	97.9
32	300	0.4	97.3	150	0.6	91.2	100	0.8	86.2	333	0.35	97.9
64	600	0.13	98.5	300	0.2	94.3	211	0.3	90.9	496	0.15	97.9
128	1200	0.05	99.2	600	0.08	96.4	432	0.09	93.0	780	0.06	97.8
256	2400	0.021	99.4	1200	0.03	98.1	875	0.04	95.2	1164	0.028	97.9
512	4799	0.003	99.6	2400	0.008	98.6	1760	0.01	97.7	1759	0.012	98.1
1024	9597	0.0003	99.6	4799	0.0007	98.2	3531	0.0009	96.6	3860	0.0008	97.9

utilization when the number of flows is small. In contrast, ABS achieves its design goal under all flow populations. Specifically, when $N \leq 32$ and utilization is the primary constraint, ABS successfully maintains link utilization at close to its target value $u^* = 98\%$. As N grows and the loss rate constraint becomes dominant, ABS is still able to effectively allocate buffer such that the average loss rate is within a close neighborhood of $p^* = 2\%$.

It is worth noting that as seen from Table II, when $N = 1024$, ABS converges the buffer size to 8613 packets. This buffer size translates into a queuing delay of 10 seconds, which is prohibitively high for most applications. However, this is not a problem of ABS, but a consequence of an unrealistic choice of p^*. In practice, router manufactures and ISPs are free to adjust u^* and p^* according to the type of service they agree to provide and the actual traffic situation. To avoid exceedingly large queuing delay, they can increase the link capacity or enforce a predetermined upper bound of buffer size to prevent queuing delay from growing beyond a certain threshold value.

We next set $N = 16$ and examine scalability of these methods to link capacities. As seen from Table III, the BDP and Stanford rules result in significant packet loss under small link capacities ($C \leq 4$ mb/s). Although they achieve both low loss rate and high utilization when C is large (e.g., $C \geq 256$ mb/s), the allocated buffer sizes are over provisioned compared to those of ABS. BSCL experiences less loss rate than the BDP and Stanford models, but it still does not lead to a buffer size that satisfies the target loss rate and utilization constraints. ABS again demonstrates the best performance among all these methods, maintaining buffer size within the target performance constraints for all link capacities.



(a) buffer size (b) loss rate

Fig. 48. ABS under changing traffic loads ($u^* = 90\%$ and $p^* = 2\%$).

3.3 Response to Load Changes

The volume of traffic perceived by any Internet router is not constant, but exhibits burstiness at different time-scales due to various reasons such as users' demand, route changes, and load balancing. Thus, stability and responsiveness in the presence of load changes is crucial for any buffer sizing scheme purported to operate in practical routers. Hence, we next examine ABS in such a scenario. We still use a "dumbbell" topology where a single bottleneck link of capacity 10 mb/s is shared by 60 heterogeneous TCP flows. The target utilization $u^* = 90\%$ and loss rate $p^* = 2\%$. As shown in Figure 48, after all flows start simultaneously at the beginning, both $b(n)$ and $p(n)$ are quickly brought to a close neighborhood of their respective stationary value. At time 48 seconds, 20 flows depart from the system. As a consequence of the reduced traffic load, packet loss rate $p(n)$ immediately drops to 1.1%, which allows the router to release memory space to meet p^* in this new scenario. After another 48 seconds, 20 more flows left and again ABS quickly shrinks the buffer. At time 144 and 192 seconds, these two sets of departed flows respectively rejoin the system and ABS is forced to increase the buffer size. It can be observed from the plots

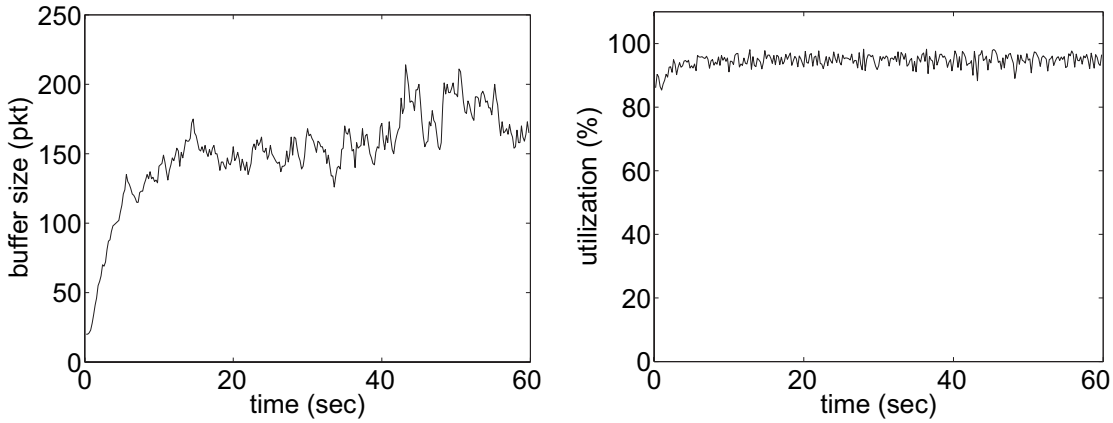
that during the entire simulation, $b(n)$ demonstrates quick responses to load changes, experiences small oscillations in both the transient and steady states, and exhibits smooth transitions between neighboring states.

3.4 Web Traffic

All scenarios considered so far have only long-lived TCP flows. However, the real Internet traffic is composed of a mixture of connections with a wide range of transfer lengths, packet sizes, and RTTs [34]. Thus, to obtain a better understanding of ABS, we next test it in more diverse scenarios.

Consider a network with a single link of capacity 10 mb/s shared by 20 persistent FTP flows in the presence of background web traffic generated by 100000 HTTP sessions. Each HTTP session downloads n_p pages with inter-page time t_p seconds, where n_p is uniformly distributed in $[10, 2000]$ and t_p is exponentially distributed with mean 1 second. Each page contains n_o objects where n_o is uniformly distributed in $[1, 5]$. The inter-object time t_o is exponentially distributed with mean 0.01 seconds. Sizes of objects follow the Pareto distribution with mean $\mu = 10$ KB and shape parameter $\alpha = 1.2$. We set the target utilization $u^* = 95\%$ and loss rate $p^* = 1\%$.

The simulation results are shown in Figure 49. As observed from Figure 49(a), ABS's behavior in this scenario differs from that of previous simulations in that the buffer size does not converge to a particular value, but fluctuates between 120 and 210 packets due to bursty ingress traffic. Note that this phenomenon does not indicate that ABS is incapable of effectively controlling short-lived web-like traffic, but actually demonstrates that this mechanism is adaptive and responsive in highly dynamic scenarios. This can be clearly seen from Figure 49(b), where utilization $p(n)$ is maintained within a close neighborhood of its target value $p^* = 95\%$.



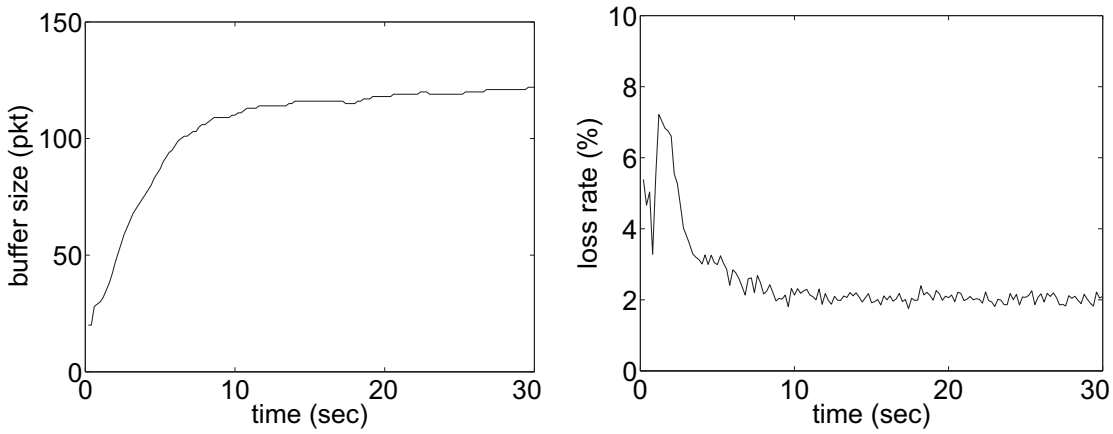
(a) buffer size (b) utilization

Fig. 49. ABS in a single link of capacity 10 mb/s shared by 20 FTP and 100000 HTTP flows ($u^* = 95\%$ and $p^* = 1\%$).

3.5 Mixture of TCP and Non-TCP Traffic

Recall that analysis of real Internet traffic traces has demonstrated that although TCP is the predominant transport protocol, a non-eligible portion of Internet traffic is contributed by non-TCP protocols [35]. Thus, in this subsection we examine ABS in a more diverse environment with 20% UDP background traffic. Consider a scenario where 20 FTP, 20 HTTP, and 20 UDP flows compete for resources of a single link of capacity 10 mb/s. Traffic parameters of HTTP flows are the same as the last subsection and each UDP flow transmits packets at a constant rate 0.1 mb/s. We set reference values of the bottleneck router to be $u^* = 90\%$ and $p^* = 2\%$. The simulation result is shown in Figure 50, where ABS is dominated by the loss constraint and $p(n)$ quickly reaches and subsequently remains in a close neighborhood of p^*.

According to Section 1, the monotonic effects of buffer size $b(n)$ on loss rate $p(n)$ and utilization $u(n)$ should also hold for traffic generated by a set of different congestion control protocols. Thus, we next test ABS under a mixture of TCP variants. Specifically, we preserve values of p^* and u^*, increase the link capacity to 100 mb/s,



(a) buffer size (b) loss rate

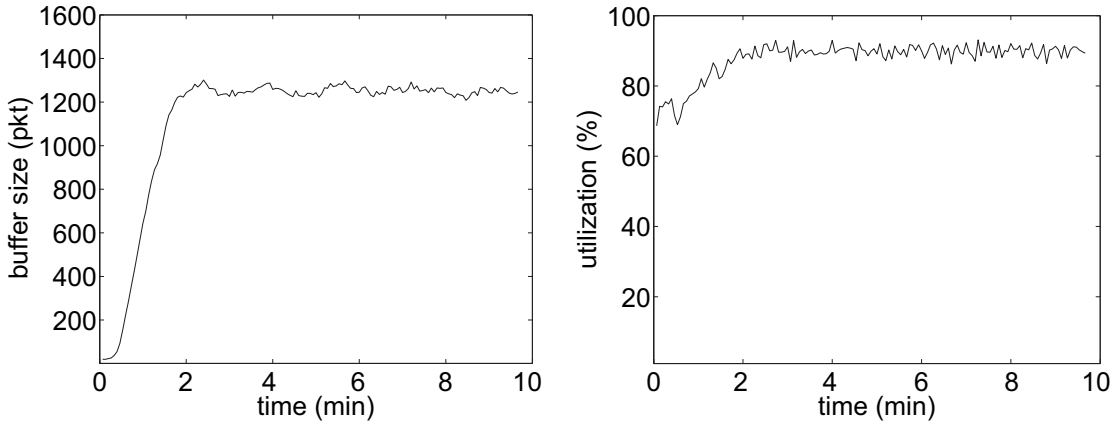
Fig. 50. ABS in a single link of capacity 10 mb/s shared by 20 FTP, 20 HTTP, and 20 UDP flows ($u^* = 90\%$ and $p^* = 2\%$).

and synthesize the ingress traffic with 10 Reno, 10 HSTCP, 10 STCP, 10 HTCP, and 10 Westwood flows with RTTs uniformly distributed within $[40, 60]$ ms. As illustrated in Figure 51, ABS successfully maintains $u(n)$ around its target value u^*.

Thus, examples provided in this and the preceding subsections clearly demonstrate ABS's excellent capability of regulating the buffer size under different traffic patterns and transport protocols, making the concept of an ABS-like dynamic scheme a highly versatile and appealing buffer sizing mechanism for real Internet routers.

3.6 Multi-Link Topology

We next extend out study to multi-link networks and see whether interactions between multiple ABS routers can produce undesirable effects. Towards this end, consider a two-link "parking lot" topology with three sets of flows. Each set is composed of 20 FTP, 10 HTTP, and 10 UDP (with constant rate 0.1 mb/s) flows. These three sets of flows respectively pass through the first link, second link, and both links. Capacities of these two links are respectively 50 and 20 mb/s. Constraint values are $u_1^* = 95\%$



(a) buffer size (b) loss rate

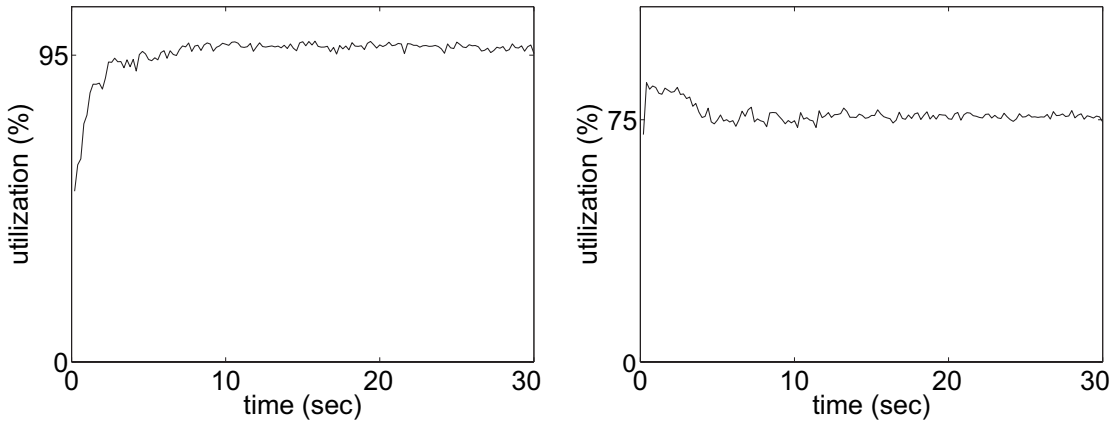
Fig. 51. ABS in a single link of capacity 100 mb/s shared by 10 Reno, 10 HSTCP, 10 STCP, 10 HTCP, and 10 Westwood flows ($u^* = 90\%$ and $p^* = 2\%$).

and $p_1^* = 1\%$ for the first link and $u_2^* = 75\%$ and $p_2^* = 5\%$ for the second link. As seen from Figure 52, two ABS routers do not intervene each other and successfully maintain utilization at their respective target level.

Based on simulations conducted in this subsection, we have demonstrated that ABS achieves its design goal – regulating buffer size $b(n)$ to satisfy the pre-specified performance constraints. Furthermore, ABS is shown to be stable in the presence of dynamically changing loads and robust to a diverse mixture of long and short TCP flows and even non-TCP traffic. All of these properties make ABS a highly appealing buffer sizing scheme for real Internet routers.

4 Discussion

In this chapter, we designed and implemented a dynamic buffer sizing scheme, called ABS, that stabilizes the buffer size to its minimum value satisfying given utilization and/or loss constraints. ABS is composed of two Integral controllers ABS_u and ABS_p, each of which is equipped with a parameter training component using a gradient-based



(a) link one (b) link two

Fig. 52. ABS in a "parking lot" topology ($p_1^* = 95\%$ and $p_2^* = 75\%$).

technique to achieve the optimal control gains. Besides stability and optimality, an appealing feature of ABS is its robustness to generic Internet traffic composed of long, short, and non-TCP flows. Thus, ABS can significantly benefit router manufactures and ISPs by improving their routers' performance, reducing system cost, and providing QoS guarantees.

We finally note that the emphasis of this chapter is not demonstrating superiority of a particular controller, but advocating a new buffer management methodology and presenting the possibility of optimally sizing router buffers using a simple yet robust controller without comprehensive knowledge of Internet dynamics. This controller actually may be replaced by more advanced candidates (e.g., nonlinear PID and variable structure control). Our future work involves designing simpler ABS-like mechanisms, analyzing ABS in its transient phase, studying its stability in more complicated congestion control models, and implementing and testing it in hardware routers under real Internet traffic.

CHAPTER VIII

SUMMARY AND FUTURE WORK

1 Summary

Our work in this dissertation can be divided into two parts. In the first part, we conducted a thorough study on designing stable congestion control protocols under heterogeneous feedback delays. We started with a preliminary investigation of the relationship between delay and stability for existing congestion control protocols proposed for the current and future Internet and concluded that none of these methods was able to offer stable performance under nontrivial end-to-end communication delays. Motivated by this finding, we set our goal to gaining a deeper understanding of the effect of delay on a system's stability and thereby designing practical congestion control methods with provable delayed stability. Towards this end, instead of focusing on a particular controller, we raised our vision to a higher horizon and sought to understand the properties of *any* control system to be stable under delay. We successfully solved this problem by finding a tight sufficient condition (whose necessity is confirmed by simulations) for any system to be stable *regardless* of delay. Employing this result and the obtained techniques, we were able to identify several sets of systems whose delayed stability were automatically established if there were stable under no delay. In addition to shedding a new light on the theoretical understanding of delay and stability, this result created an avenue to developing new congestion control systems whose stability was *independent* of delay.

As applications of this method, we proposed a series of congestion control mechanisms that offered delay-independent stability as well as many other appealing prop-

erties. Specifically, we first introduced three novel modifications to the classic Kelly control such that the Jacobian matrix of the resulting system satisfied the condition of delay-independent stability. We called this new controller Max-min Kelly Control (MKC) and demonstrated that MKC achieved stability under heterogeneous time-varying delay, converged to efficient link utilization exponentially fast, were free from oscillations in both the steady state and transient phase, and admitted a low-overhead implementation. MKC has received a fair amount of attention from the research community since its introduction in 2004. For instance, it has been extended by Miller *et al.* [79] in their development of two new congestion control schemes, eXtended MKC (XMKC) and Utility MKC (UMKC). However, we further identified that MKC suffered from persistent packet loss in the equilibrium and slow convergence rate to fair resource allocation. We successfully overcome these two drawbacks by proposing a new congestion control protocol called JetMax, which achieved zero packet loss during the entire lifetime of a flow, constant convergence time to both efficiency and fairness, and delay-independent stability. These properties have been confirmed by our extensive `ns2` simulations and Linux experiments.

In the second part of this dissertation, we sought to address the problem of optimal buffer sizing. Our motivation was driven by the fact that all existing buffer sizing rules were based a simplistic single-TCP model, which did not capture the characteristics of traffic encountered by routers in the real Internet. As a consequence, these methods were subject to over- or under-provisioning of router buffers, leading to unacceptably long queue lengths (and therefore queuing delays) or high packet lost rates. This problem cannot be solved by traditional AQM methods (e.g., RED, PI, and REM). This is because these methods aimed to stabilize the queuing delay (or queue length) at certain target value and would exhibit unpredictable behavior and became simply unstable when this target value was above the provisioned buffer

size. This problem is further complicated by the fact that the required buffer size is a time-varying metric as the traffic load changes over time. A natural remedy to this problem is to dynamically adapt the buffer size according its current incoming traffic.

Towards this end, we proposed a buffer management scheme called Adaptive Buffer Sizing (ABS), which based on dynamics of the input traffic adaptively allocated the optimal amount of buffer under given performance constraints, utilization and packet loss. ABS is composed of two sub-system ABS_u and ABS_p for the utilization and loss constraints, respectively. Each sub-system is a PI controller, whose gain parameters are intelligently tuned using a gradient-based controller. This way, these two sub-controllers are able to adaptively set their control gains to their optimal values under dynamically changing incoming traffic. We also introduced additional mechanisms to differentiate bottleneck and non-bottleneck routers and balance the tradeoff between packet loss and delay. Our extensive `ns2` simulations demonstrated that ABS were robust to generic Internet traffic, scalable to a large flow population, and responsive to traffic load changes.

2 Future Work

There are several problems remaining open in this work. First, as identified in our comparison study of existing explicit congestion control methods, XCP is subject to the bottleneck oscillations in certain multi-link topologies. As we mentioned earlier, this problem applied to all congestion control protocols that relied on single-link feedback. Even though we could not simulation scenarios in which MKC or JetMax exhibited bottleneck oscillations and became unstable, we at the same time could not provide a rigorous proof of their stability/instability in such cases. Therefore, our future work involves developing a generic framework for modeling the bottleneck

oscillation problem and understanding the properties required by a congestion control protocol to be stable in multi-link topologies. Our ultimate goal is to design a new protocol that satisfies all properties required by an ideal congestion, i.e., topology-independent, delay-independent stability, exponential convergence to both efficiency and fairness, and zero packet loss in both the steady state and transient phase.

In our future work in the area of buffer sizing, we plan to implement ABS in Linux routers and conduct a comprehensive experimental study of it. We are also interested in investigating the result of coupling ABS with other traditional AQM methods and developing mechanisms that can be utilized by end-users to take better advantage of ABS. Exploring the possibility of incorporating ABS into the framework of PERT [12] forms another line of my future research.

REFERENCES

[1] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," *IETF RFC 2581*, Apr. 1999.

[2] H. C. Andersen, A. Lotfi, and A. C. Tsoi, "A New Approach to Adaptive Fuzzy Control: The Controller Output Error Method," *IEEE Trans. Syst., Man, Cybern.*, vol. 27, no. 4, pp. 686–691, Aug. 1997.

[3] G. Appenseller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," in *Proc. ACM SIGCOMM*, Aug. 2004, pp. 281–292.

[4] S. Athuraliya, S. H. Low, V. H. Li, and Q. Yin, "REM: Active Queue Management," *IEEE Networks*, vol. 15, no. 3, pp. 48–53, Jun. 2001.

[5] K. Avrachenkov, U. Ayesta, and A. Piunovskiy, "Optimal Choice of the Buffer Size in the Internet Routers," in *Proc. IEEE CDC*, Dec. 2005, pp. 1143–1148.

[6] J. Aweya, M. Ouellette, and D. Y. Dontuno, "A Simple, Scalable and Provably Stable Explicit Rate Computation Scheme for Flow Control in Communication Networks," *Int. J. of Comm. Systems*, vol. 14, no. 6, pp. 593–618, Jun. 2001.

[7] F. L. Bauer, J. Stoer, and C. Witzgall, "Absolute and Monotonic Norms," *Numerische Mathematik*, vol. 3, no. 1, pp. 257–264, 1961.

[8] N. Beheshti, Y. Ganjali, R. Rajaduray, D. Blumenthal, and N. McKeown, "Buffer Sizing in All-Optical Packet Switches," in *Proc. OFC/NFOEC*, Mar. 2006.

[9] D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, N.J.: Prentice-Hall, 1992.

[10] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Englewood Cliffs, N.J.: Prentice Hall, 1989.

[11] S. Bhandarkar, S. Jain, and A. L. N. Reddy, "LTCP: Improving the Performance of TCP in Highspeed Networks," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 36, no. 1, Jan. 2006.

[12] S. Bhandarkar, A. L. N. Reddy, Y. Zhang, and D. Loguinov, "Emulating AQM from End Hosts," in *Proc. ACM SIGCOMM*, Aug. 2007, pp. 349–360.

[13] I. Bisio and M. Marchese, "Analytical Expression and Performance Evaluation of TCP Packet Loss Probability over Geostationary Satellite," *IEEE Commun. Lett.*, vol. 8, no. 4, pp. 232–234, Apr. 2004.

[14] F. Blanchini, R. L. Cigno, and R. Tempo, "Robust Rate Control for Integrated Services Packet Networks," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 644–652, 2002.

[15] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," in *Proc. ACM SIGCOMM*, Aug. 1994, pp. 24–35.

[16] R. Bronson, *Schaum's Outline of Theory and Problems of Matrix Operations*. New York: McGraw-Hill, 1989.

[17] J. Cao, W. S. Cleveland, D. Lin, and D. X. Sun, "Internet Traffic Tends Toward Poisson and Independent as the Load Increases," in *Nonlinear Estimation and Classification*. New York: Springer-Verlag, 2003.

[18] D. Chazan and W. L. Miranker, "Chaotic Relaxation," *Linear Alg. Appl.*, vol. 2, pp. 199–222, 1969.

[19] D.-M. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for

Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, vol. 17, no. 1, pp. 1–14, Jun. 1989.

[20] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith, "Tuning RED for Web Traffic," in *Proc. ACM SIGCOMM*, Aug. 2000, pp. 139–150.

[21] M. Dai and D. Loguinov, "Analysis of Rate-Distortion Functions and Congestion Control in Scalable Internet Video Streaming," in *Proc. ACM NOSSDAV*, Jun. 2003, pp. 60–69.

[22] S. Deb and R. Srikant, "Global Stability of Congestion Controllers for the Internet," *IEEE Trans. Automat. Contr.*, vol. 48, no. 6, pp. 1055–1060, Jun. 2003.

[23] S. Deb and R. Srikant, "Rate-Based versus Queue-Based Models of Congestion Control," in *Proc. ACM SIGMETRICS*, Jun. 2004, pp. 246–257.

[24] R. DeCarlo, M. S. Branicky, S. Pettersson, and B. Lennartson, "Perspectives and Results on the Stability and Stabilizability of Hybrid Systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 1069–1082, Jul. 2000.

[25] A. Dhamdhere, H. Jiang, and C. Dovrolis, "Buffer Sizing for Congested Internet Links," in *Proc. IEEE INFOCOM*, Mar. 2005, pp. 1072–1083.

[26] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor Sharing Flows in the Internet," in *Proc. IEEE IWQoS*, Jun. 2005.

[27] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, "Routers with Very Small Buffers," in *Proc. IEEE INFOCOM*, Apr. 2006.

[28] D. Y. Eun and X. Wang, "Performance Analysis of TCP/AQM with Generalized AIMD under Intermediate Buffer Sizes," in *Proc. IEEE IPCCC*, Apr. 2006, pp. 367–374.

[29] A. Falk and D. Katabi, "Specification for the Explicit Control Protocol (XCP)," USC/ISI, Tech. Rep., Oct. 2005.

[30] S. Floyd, "High-Speed TCP for Large Congestion Windows," *IETF RFC 3649*, Dec. 2003.

[31] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," in *Proc. ACM SIGCOMM*, Aug. 2000, pp. 43–56.

[32] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," *IETF RFC 2582*, Apr. 1999.

[33] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[34] S. Floyd and E. Kohler, "Internet Research Needs Better Models," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 33, no. 1, pp. 29–34, Jan. 2003.

[35] M. Fomenkov, K. Keys, D. Moore, and K. Claffy, "Longitudinal Study of Internet Traffic from 1998-2003," in *Proc. WISICT*, Jan. 2004, pp. 1–6.

[36] C. Fulton, S.-Q. Li, and C. S. Lim, "An ABR Feedback Control Scheme with Tracking," in *Proc. IEEE INFOCOM*, Apr. 1997, pp. 805–814.

[37] Y. Ganjali and N. McKeown, "Update on Buffer Sizing in Internet Routers," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 36, no. 5, pp. 67–70, Oct. 2006.

[38] M. Gerla, M. Y. Sanadidi, R. Wang, A. Zanella, C. Casetti, and S. Mascolo, "TCP Westwood: Congestion Window Control Using Bandwidth Estimation," in *Proc. IEEE GLOBECOM*, Nov. 2001, pp. 1698–1702.

[39] P. Glendinning, *Stability, Instability and Chaos: An Introduction to the Theory of Nonlinear Differential Equations*. New York: Cambridge University Press,

1994.

[40] W.-B. Gong and S. Nananukul, "Rational Interpolation for Rare Event Probabilities," in *Stochastic Networks: Stability and Rare Events*. New York: Springer-Verlag, 1996.

[41] S. Gorinsky, A. Kantawala, and J. Turner, "Link Buffer Sizing: A New Look at the Old Problem," in *Proc. IEEE ISCC*, Jun. 2005, pp. 507–514.

[42] S. Gorinsky, A. Kantawala, and J. Turner, "Simulation Perspectives on Link Buffer Sizing," *Simulation*, vol. 83, no. 3, pp. 245–257, 2007.

[43] C. V. Hollot, V. Misra, D. Towsley, and W. Gong, "Analysis and Design of Controllers for AQM Routers Supporting TCP Flows," *IEEE Trans. Automat. Contr.*, vol. 47, no. 6, pp. 945–959, Jun. 2002.

[44] C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "On Designing Improved Controllers for AQM Routers Supporting TCP Flows," in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 1726–1734.

[45] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge / New York: Cambridge University Press, 1985.

[46] F. Ishizaki and T. Takine, "Loss Probability in a Finite Discrete-Time Queue in Terms of the Steady State Distribution of an Infinite Queue," *Queueing Systems: Theory and Applications*, vol. 31, no. 3-4, pp. 317–326, Mar. 1999.

[47] V. Jacobson, "Congestion Avoidance and Control," in *Proc. ACM SIGCOMM*, Aug. 1988, pp. 314–329.

[48] JetMax@TAMU. [Online]. Available: http://irl.cs.tamu.edu/projects/mkc/. Accessed on Feb. 19, 2008.

[49] C. Jin, D. Wei, and S. H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," in *Proc. IEEE INFOCOM*, Mar. 2004, pp. 2490–2501.

[50] C. Jin, D. X. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S. Singh, "FAST TCP: From Theory to Experiments," *IEEE Network*, vol. 19, no. 1, pp. 4–11, Jan. 2005.

[51] R. Johari and D. K. H. Tan, "End-to-End Congestion Control for the Internet: Delays and Stability," *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 818–832, Dec. 2001.

[52] K. Kar, S. Sarkar, and L. Tassiulas, "A Simple Rate Control Algorithm for Maximizing Total User Utility," in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 133–141.

[53] E. Kaszkurewicz and A. Bhaya, "A Delay-Independent Robust Stability Condition for Linear Discrete-Time Systems," in *Proc. IEEE CDC*, Dec. 1993, pp. 3459–3460.

[54] E. Kaszkurewicz and A. Bhaya, *Matrix Diagonal Stability in Systems and Computation.* Boston: Birkhäuser, 2000.

[55] E. Kaszkurewicz, A. Bhaya, and D. D. Šiljak, "On the Convergence of Parallel Asynchronous Block-Iterative Computations," *Linear Alg. Appl.*, vol. 131, pp. 139–160, 1990.

[56] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth Delay Product Networks," in *Proc. ACM SIGCOMM*, Aug. 2002, pp. 89–102.

[57] C. Kellett, R. Shorten, and D. J. Leith, "Sizing Internet Router Buffers, Active Queue Management, and the Lur'e Problem," in *Proc. IEEE CDC*, Dec. 2006,

pp. 650–654.

[58] W. G. Kelley and A. C. Peterson, *Difference Equations: An Introduction with Applications*. San Diego / London: Academic Press, 2001.

[59] F. P. Kelly, "Charging and Rate Control for Elastic Traffic," *Euro. Trans. on Telecommun.*, vol. 8, no. 1, pp. 33–37, Jan. 1997.

[60] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability," *J. of Oper. Res. Soc.*, vol. 49, no. 3, pp. 237–252, Mar. 1998.

[61] T. Kelly, "Scalable TCP: Improving Performance in High-Speed Wide Area Networks," *Computer Communication Review*, vol. 33, no. 2, pp. 83–91, Apr. 2003.

[62] S. Kunniyur, "AntiECN Marking: A Marking Scheme for High Bandwidth Delay Connections," in *Proc. IEEE ICC*, May 2003, pp. 647–651.

[63] S. Kunniyur and R. Srikant, "A Time-Scale Decomposition Approach to Adaptive Explicit Congestion Notification (ECN) Marking," *IEEE Trans. Automat. Contr.*, vol. 47, no. 6, pp. 882–894, Jun. 2002.

[64] S. Kunniyur and R. Srikant, "End-to-End Congestion Control Schemes: Utility Functions, Random Losses and ECN Marks," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 689–702, Oct. 2003.

[65] S. Kunniyur and R. Srikant, "Stable, Scalable, Fair Congestion Control and AQM Schemes That Achieve High Utilization in the Internet," *IEEE Trans. Automat. Contr.*, vol. 48, no. 11, pp. 2024–2029, Nov. 2003.

[66] S. Kunniyur and R. Srikant, "Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management," in *Proc. ACM SIGCOMM*,

Aug. 2001, pp. 123–134.

[67] A. Kuzmanovic and E. Knightly, "TCP-LP: A Distributed Algorithm for Low Priority Data Transfer," in *Proc. IEEE INFOCOM*, Apr. 2003, pp. 1691–1701.

[68] A. Lakshmikantha, R. Srikant, and C. Beck, "Impact of File Arrivals and Departures on Buffer Sizing in Core Routers," in *Proc. IEEE INFOCOM*, Apr. 2008.

[69] D. Leith and R. Shorten, "H-TCP Protocol for High-Speed Long Distance Networks," in *Proc. PFLDnet*, Feb. 2004.

[70] N. Likhanov, B. Tsybakov, and N. Georganas, "Analysis of an ATM Buffer with Self-Similar ("fractal") Input Traffic," in *Proc. IEEE INFOCOM*, Apr. 1995, pp. 985–992.

[71] S. Liu, T. Basar, and R. Srikant, "Pitfalls in the Fluid Modeling of RTT Variations in Window-Based Congestion Control," in *Proc. IEEE INFOCOM*, Mar. 2005, pp. 1002–1012.

[72] D. Loguinov and H. Radha, "End-to-End Rate-Based Congestion Control: Convergence Properties and Salability Analysis," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 564–577, Aug. 2003.

[73] S. H. Low, "A Duality Model of TCP and Queue Management Algorithms," *IEEE/ACM Trans. Netw.*, vol. 11, no. 4, pp. 525–536, Aug. 2003.

[74] S. H. Low, L. L. H. Andrew, and B. P. Wydrowski, "Understanding XCP: Equilibrium and Fairness," in *Proc. IEEE INFOCOM*, Mar. 2005, pp. 1025–1036.

[75] S. H. Low and D. E. Lapsley, "Optimization Flow Control I: Basic Algorithm and Convergence," *IEEE/ACM Trans. Netw.*, vol. 7, no. 6, pp. 861–874, Dec.

1999.

[76] S. H. Low, F. Paganini, J. Wang, S. Adlakha, and J. Doyle, "Dynamics of TCP/RED and a Scalable Control," in *Proc. IEEE INFOCOM*, Jun. 2002, pp. 239–248.

[77] L. Massoulié, "Stability of Distributed Congestion Control with Heterogeneous Feedback Delays," *IEEE Trans. Automat. Contr.*, vol. 47, no. 6, pp. 895–902, Jun. 2002.

[78] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 27, no. 3, pp. 67–82, Jul. 1997.

[79] K. Miller and T. Harks, "Utility Max-Min Fair Congestion Control with Time-Varying Delays," in *Proc. IEEE INFOCOM*, Apr. 2008.

[80] T. Mori, N. Fukuma, and M. Kuwahara, "Delay-Independent Stability Criteria for Discrete-Delay Systems," *IEEE Trans. Automat. Contr.*, vol. 27, no. 4, pp. 964–966, Aug. 1982.

[81] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and Its Empirical Validation," in *Proc. ACM SIGCOMM*, Sep. 1998, pp. 303–314.

[82] F. Paganini, J. Doyle, and S. H. Low, *A Control Theoretical Look at Internet Congestion Control*, ser. Multidisciplinary Research in Control: The Mohammed Dahleh Symposium 2002. Berlin / Heidelberg: Springer, 2003.

[83] V. Paxson, "Empirically Derived Analytic Models of Wide-Area TCP Connections," *IEEE/ACM Trans. Netw.*, vol. 2, no. 4, pp. 316–328, Aug. 1994.

[84] R. S. Prasad, M. Jain, and C. Dovrolis, "On the Effectiveness of Delay-Based

Congestion Avoidance," in *Proc. PFLDnet*, Feb. 2004.

[85] R. Prasad, C. Dovrolis, and M. Thottan, "Router Buffer Sizing Revisited: The Role of the Output/Input Capacity Ratio," in *Proc. ACM SIGCOMM CoNEXT*, Dec. 2007.

[86] V. V. Prasolov, *Problems and Theorems in Linear Algebra.* Providence, R.I.: American Mathematical Society, 1994.

[87] G. Raina, D. Towsley, and D. Wischik, "Part II: Control Theory for Buffer Sizing," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 35, no. 3, pp. 79–82, Jul. 2005.

[88] K. K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," *IETF RFC 3168*, Sep. 2001.

[89] I. Rhee and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," in *Proc. PFLDnet*, Feb. 2005.

[90] T. G. Robertazzi, *Computer Networks and Systems: Queuing Theory and Performance Evaluation.* New York: Springer-Verlag, 1994.

[91] Cisco 3600 Series Routers. [Online]. Available: http://www.cisco.com/en/US/products/hw/routers/ps274/. Accessed on Feb. 19, 2008.

[92] R. N. Shorten and D. J. Leith, "On Queue Provisioning, Network Efficiency and the Transmission Control Protocol," Hamilton Institute, Tech. Rep., 2006.

[93] J. A. Snyman, *Practical Mathematical Optimization : An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms.* New York: Springer, 2005.

[94] J. Sommers, P. Barford, A. Greenberg, and W. Willinger, "An SLA Perspective on the Router Buffer Sizing Problem," *ACM SIGMETRICS Performance*

Evaluation Review, vol. 35, no. 4, Mar. 2008.

[95] Y. Su, A. Bhaya, E. Kaszkurewicz, and V. S. Kozyakjin, "Further Results on Stability of Asynchronous Discrete-time Linear Systems," in *Proc. IEEE CDC*, Dec. 1997, pp. 915–920.

[96] C. Villamizar and C. Song, "High Performance TCP in the ANSNET," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 24, no. 5, pp. 45–60, Oct. 1994.

[97] G. Vinnicombe, "On the Stability of End-to-End Congestion Control for the Internet," Cambridge University, Tech. Rep. CUED/F-INFENG/TR.398, Dec. 2000.

[98] G. Vinnicombe, "On the Stability of Networks Operating TCP-like Protocols," in *Proc. IFAC*, Aug. 2002.

[99] G. Vinnicombe, "Robust Congestion Control for the Internet," Cambridge University, Tech. Rep., 2002.

[100] J. Wang, D. X. Wei, and S. H. Low, "Modeling and Stability of FAST TCP," in *Proc. IEEE INFOCOM*, Mar. 2005, pp. 938–948.

[101] D. X. Wei and P. Cao, "NS-2 TCP-Linux: An NS-2 TCP Implementation with Congestion Control Algorithms from Linux," in *Proc. WNS2*, Oct. 2006, pp. 9–18.

[102] D. Wischik, "Buffer Requirements for High-Speed Routers," in *Proc. ECOC*, Sep. 2005, pp. 23–26.

[103] D. Wischik and N. McKeown, "Part I: Buffer Sizes for Core Routers," *ACM SIGCOMM Comp. Comm. Rev.*, vol. 35, no. 2, pp. 75–78, Jul. 2005.

[104] M. K. Wong and F. Bonomi, "A Novel Explicit Rate Congestion Control Algorithm," in *Proc. IEEE GLOBECOM*, Nov. 1998, pp. 8–12.

[105] J. Wu and H. El-Ocla, "TCP Congestion Avoidance Model with Congestive Loss," in *Proc. IEEE ICON*, Nov. 2004, pp. 3–8.

[106] B. P. Wydrowski, L. L. H. Andrew, and I. M. Y. Mareels, "MaxNet: Faster Flow Control Convergence," *Networking*, vol. 3042, pp. 588–599, May 2004.

[107] B. P. Wydrowski and M. Zukerman, "MaxNet: A Congestion Control Architecture for Maxmin Fairness," *IEEE Commun. Lett.*, vol. 6, no. 11, pp. 588–599, Nov. 2002.

[108] XCP@ISI. [Online]. Available: http://www.isi.edu/isi-xcp/. Accessed on Feb. 19, 2008.

[109] Y. Xiong, J.-C. Liu, K. G. Shin, and W. Zhao, "On the Modeling and Optimization of Discontinuous Network Congestion Control Systems," in *Proc. IEEE INFOCOM*, Mar. 2004, pp. 2812–2820.

[110] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast, Long Distance Networks," in *Proc. IEEE INFOCOM*, Mar. 2004, pp. 2514–2524.

[111] Y. R. Yang and S. S. Lam, "General AIMD Congestion Control," in *Proc. IEEE ICNP*, Nov. 2000, pp. 187–198.

[112] L. Ying, G. E. Dullerud, and R. Srikant, "Global Stability of Internet Congestion Control with Heterogeneous Delays," *IEEE/ACM Trans. Netw.*, vol. 14, no. 3, pp. 579–590, Jun. 2006.

[113] C.-C. Yu, *Autotuning of PID Controllers: A Relay Feedback Approach*. London: Springer, 2006.

[114] Y. Zhang and T. Henderson, "An Implementation and Experimental Study of the eXplicit Control Protocol (XCP)," in *Proc. IEEE INFOCOM*, Mar. 2005,

pp. 1037–1048.

[115] Y. Zhang, S.-R. Kang, and D. Loguinov, "Delayed Stability and Performance of Distributed Congestion Control," in *Proc. ACM SIGCOMM*, Aug. 2004, pp. 307–318.

[116] Y. Zhang, D. Leonard, and D. Loguinov, "JetMax: Scalable Maxmin Congestion Control for High-Speed Heterogeneous Networks," in *Proc. IEEE INFOCOM*, Apr. 2006.

[117] Y. Zhang and D. Loguinov, "Delay-Independent Diagonal Stability of Single-Link Congestion Control," in *Proc. IEEE CDC*, Dec. 2006, pp. 621–626.

VITA

Yueping Zhang received a B.S. degree in Computer Science from Beijing University of Aeronautics and Astronautics in July 2001 and graduated with his Ph.D. in Computer Engineering from Texas A&M University (TAMU) in May 2008.

He joined the Internet Research Lab (IRL) at TAMU in May 2003. His research interests include congestion control, delayed stability analysis, active queue management (AQM), router buffer sizing, and peer-to-peer networks. He can be reached at:

Yueping Zhang C/O Minglong Zhang

2 Jinhai Ave.

Zhengzhou, Henan 450015, China

The typist for this dissertation was Yueping Zhang.