Available online at www.sciencedirect.com

ScienceDirect

Computer Networks 52 (2008) 1193–1219

Computer Networks

www.elsevier.com/locate/comnet

# Jetmax: Scalable max–min congestion control for high-speed heterogeneous networks ☆

Yueping Zhang *, Derek Leonard, Dmitri Loguinov

*Department of Computer Science, Texas A&M University, College Station, TX 77843, United States*

Received 11 March 2007; received in revised form 24 October 2007; accepted 12 November 2007
Available online 18 January 2008

Responsible Editor: Marco Ajmone Marsan

**Abstract**

Recent surge of interest towards congestion control that relies on *single-link* feedback (e.g., XCP, RCP, MaxNet, EMKC, VCP), suggests that such systems may offer certain benefits over traditional models of additive packet loss. Besides topology-independent stability and faster convergence to efficiency/fairness, it was recently shown that any stable single-link system with a symmetric Jacobian tolerates arbitrary fixed, as well as *time-varying*, feedback delays. Although delay-independence is an appealing characteristic, the EMKC system developed in exhibits undesirable equilibrium properties and slow convergence behavior. To overcome these drawbacks, we propose a new method called JetMax and show that it admits a low-overhead implementation inside routers (three additions per packet), overshoot-free transient and steady state, tunable link utilization, and delay-insensitive flow dynamics. The proposed framework also provides capacity-independent convergence time, where fairness and utilization are reached in the same number of RTT steps for a link of *any* bandwidth. Given a 1 mb/s, 10 gb/s, or googol ($10^{100}$) bps link, the method converges to within 1% of the stationary state in six RTTs. We finish the paper by comparing JetMax's performance to that of existing methods in `ns2` simulations and discussing its Linux implementation.
© 2008 Elsevier B.V. All rights reserved.

*Keywords:* Congestion control; Multi-link stability; Max–min fairness; High-speed networks

## 1. Introduction

In the light of TCP's scalability issues in high-speed networks [9], explicit-feedback congestion control has gained renewed interest in the last several years [16,22,32,33]. 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 [16], packet loss [33], single-bit congestion indication [12,18,26], queuing delay [14,29], or link prices [17,20,23]. This information helps end-flows converge their sending rates to some social optimum and achieve a certain optimization objective.

☆ An earlier version of this paper has appeared in IEEE INFOCOM 2006.
* Corresponding author. Tel.: +1 979 862 9347; fax: +1 419 831 9347.
*E-mail addresses:* yueping@cs.tamu.edu (Y. Zhang), dleonard@cs.tamu.edu (D. Leonard), dmitri@cs.tamu.edu (D. Loguinov).

1389-1286/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2007.11.021

Unlike some of the largely ineffective AQM aimed at improving the performance of TCP [4], 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 paper 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.

The first half of the paper deals with understanding delayed stability and convergence performance of several recently proposed AQM approaches: eXplicit Control Protocol (XCP) [16], Rate Control Protocol (RCP) [7], Exponential Max–min Kelly Control (EMKC) [33], and a hybrid method suggested in [33] that combines EMKC with Adaptive Virtual Queue (AVQ) [20,19]. 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 [7,16,29,33]. 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 paper 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 the paper 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.

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

- *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 paper 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.

The rest of the paper is organized as follows. We review the existing explicit congestion control algorithms in Section 2 and identify their problems in Section 3. We then highlight the importance of studying multi-link stability of max–min systems in Section 4. Following that, we introduce JetMax in Section 5 and discuss its implementation issues in Section 6. We then demonstrate JetMax's performance through `ns2` simulations and Linux experiments in Sections 7 and 8, respectively. We conclude the paper in Section 9.

## 2. Background

We start by describing the notation used throughout the paper. Assume $N$ users in the network whose rates at time $t$ are given by $\{x_r(t)\}_{r=1}^{N}$. Following notation in [15], we denote the RTT of each flow by $D_r(t)$ and the forward/backward delays of user $r$ to/from link $l$ by $D^{\rightarrow}_{r,l}(t)$ and $D^{\leftarrow}_{r,l}(t)$, 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 [16] has become a de-facto standard for explicit congestion control in IP networks [8]. 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 $\alpha$ and $\beta$ are constants used in the XCP control equation.

XCP's design goals [16] include max–min fairness and high link utilization; however, a recent study of its equilibrium properties [22] 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 [22]. Another study [32] 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) [7] 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

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

where $\Delta$ is the router's control interval, $\alpha$ and $\beta$ 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 [7].

Two additional max–min methods are inspired by Kelly's optimization framework [17] and aim to improve stability and convergence properties of traditional models of additive packet loss [19,23]. The first approach called MaxNet [29] 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

$$\dot{p}_l(t) = \frac{y_l(t) - \gamma C_l}{C_l}, \tag{2}$$

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

The second method is Exponential Max–min Kelly Control (EMKC) [33], 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

$$x_r(n) = x_r(n - D_r) + \alpha - \beta p_r(n) x_r(n - D_r), \tag{3}$$

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 [33]

$$p_r(n) = \max_{l \in r} \frac{\sum_{s \in l} x_s(n - D^{\rightarrow}_{s,l} - D^{\leftarrow}_{r,l}) - C_l}{\sum_{s \in l} x_s(n - D^{\rightarrow}_{s,l} - D^{\leftarrow}_{r,l})}. \tag{4}$$

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

For a single-link network, systems (3) and (4) are locally asymptotically stable for all time-varying delays. Due to the steady-state overshoot of link's capacity [33], EMKC does not reach max–min fairness. However, as suggested in [33], EMKC can be combined with AVQ [20] 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 [25]. Traditional models of congestion control [16,19,23,27] 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 [21]. It thus becomes important to examine how protocols behave in highly heterogeneous environments and whether fluctuating feedback delay may cause them to oscillate.

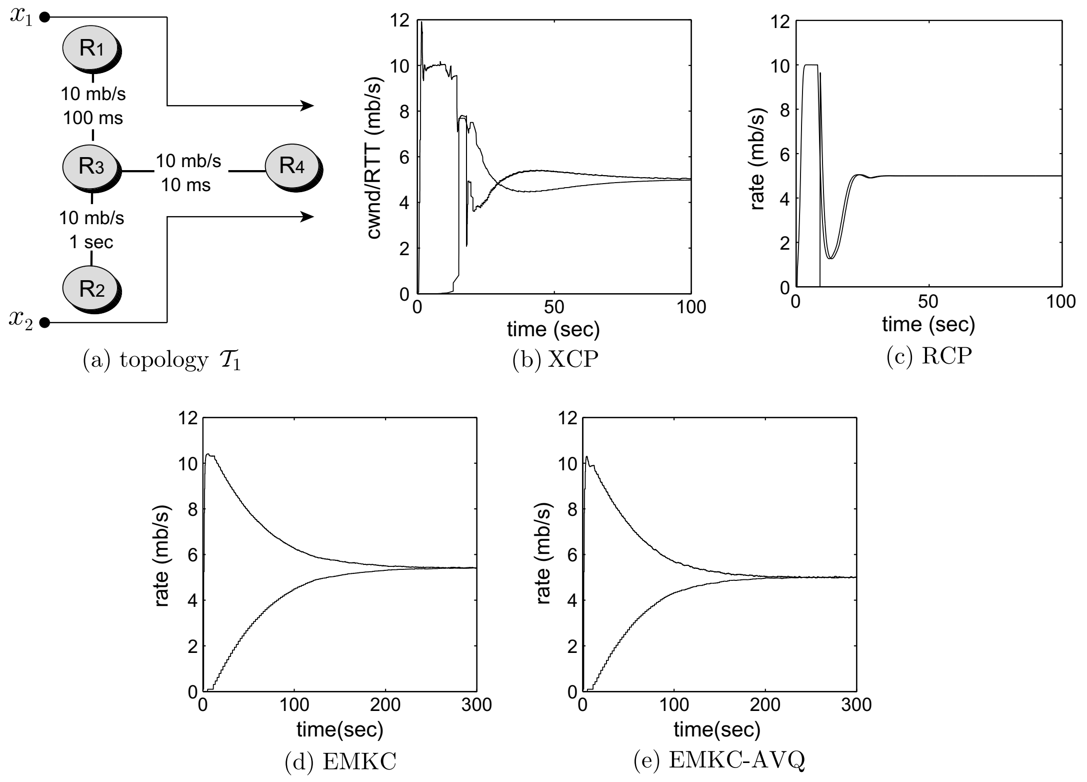
Besides stability, ideal congestion control should exhibit fast convergence to both efficiency and fairness, avoid overshooting capacity in transient and stationary states, and converge to the desired link utilization $\gamma$. While the first few factors are mostly important to end-users, the last metric is of interest to network operators, who usually run their backbones at well below capacity and may not appreciate protocols (such as [14,16]) 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 [16], RCP [7], and EMKC [33] as completely different approaches to max–min congestion control. At the time of this writing, MaxNet [29] did not have a publicly available `ns2` implementation; however, we found that a combination of EMKC and AVQ [20] 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 $\gamma C_l$, where $\gamma$ is the desired link utilization. This method is similar

[2] Since all MKC-based methods examined in the paper 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.



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

(d) EMKC (e) EMKC-AVQ

Fig. 1. XCP, RCP, EMKC, and EMKC-AVQ in topology $\mathcal{T}_1$.

to the price integrator (2) 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 [16,7,33], respectively. We also experimented with the modified XCP code from ISI [30] and found it to offer no stability benefits over the original code. We thus limit our XCP discussion to the algorithms used in [16].

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. 1a, where two flows $x_1$ and $x_2$ with round-trip delays 220 and 2020 ms, respectively, start with a 5-s delay and share a 10-mb/s link. For XCP, we use the parameters suggested in [16] (i.e., $\alpha = 0.4$ and $\beta = 0.226$) and set the buffer size sufficiently large (i.e., at least $C_l \times \text{RTT}$). As Fig. 1c 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. 1c. 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

[3] For all rate-based methods examined in the paper (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.

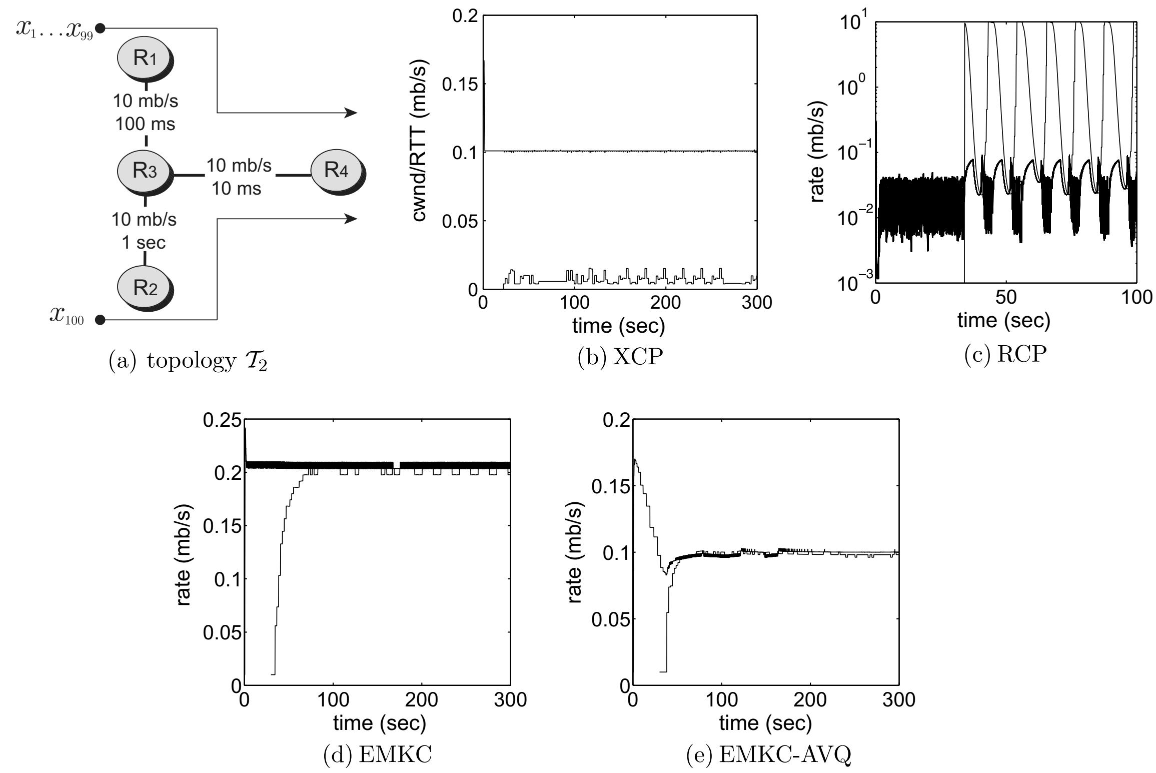


Fig. 2. XCP, RCP, EMKC, and EMKC-AVQ in topology $\mathcal{T}_2$.

$\mathcal{T}_1$. The result is plotted in Fig. 1d, which demonstrates that EMKC converges to the stationary state much more smoothly than XCP and RCP; however, it spends over 250 s before reaching fairness and eventually overshoots link's capacity by 8%. Although EMKC's convergence rate can be improved by increasing $\alpha$, this leads to more steady-state packet loss and larger overshoot [33]. 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 state, we modified AVQ to estimate the aggregate input rate $y_l(n)$ every $\Delta$ time units and adjust the virtual capacity $\widetilde{C}_l$ at the end of this interval:

$$\widetilde{C}_l(n) = \widetilde{C}_l(n-\Delta) + \frac{\tau\Delta(\gamma C_l - y_l(n))}{D_{\max}}, \quad (5)$$

where $\tau = 0.2$ is the gain parameter used throughout this paper, $\gamma$ 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 (5) is in fact an Integral controller [31] on virtual capacity $\widetilde{C}_l(n)$ and converges combined rate $y_l(n)$ to its target value $\gamma C_l$. The final step of EMKC-AVQ is to limit $\widetilde{C}_l$ to the range $(-\infty, \gamma C_l]$ and then apply its value in (4) to compute the feedback. Using this implementation, we repeat the above simulation and plot the result in Fig. 1e, 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 s).

### 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. 2.

[4] All delays are computed using XCP's smoothed EWMA estimator with the default weight 0.4 and (5) is normalized by $D_{\max}$ to ensure stability of the resulting system under delayed feedback [19].

As seen from Fig. 2d and 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* 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-s delay. This results in unstable oscillations shown in Fig. 2b and 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. 3a, 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 $[5000, 10\,000]$, 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. 3, 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

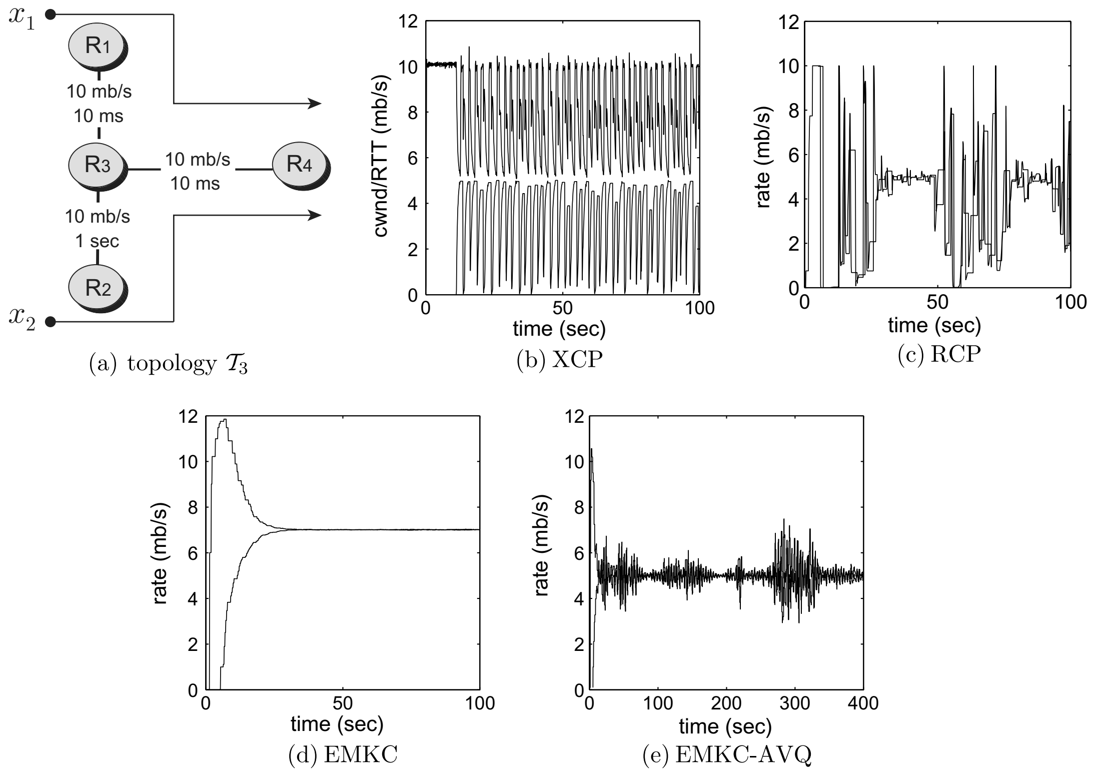


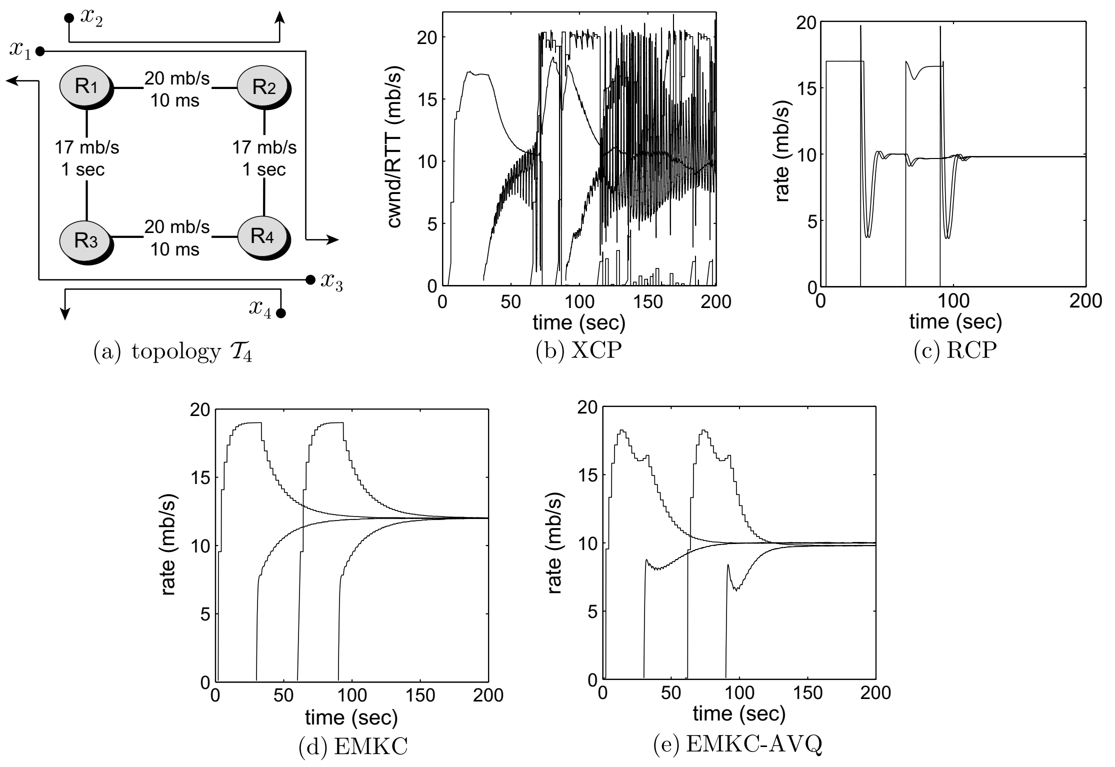
Fig. 3. XCP, RCP, EMKC, and EMKC-AVQ in topology $\mathcal{T}_3$.

EMKC-AVQ (all of which use $D_{\max}$ in the 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 $\mathscr{T}_4$ shown in Fig. 4a, 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-s 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. 4b shows the behavior of XCP in $\mathscr{T}_4$. Notice in the figure that the protocol not only oscillates for over 200 s, 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. 4c, EMKC in Fig. 4d, and EMKC-AVQ in Fig. 4e have no visible stability problems and converge their sending rates exactly as expected.



(a) topology $\mathcal{T}_4$ (b) XCP (c) RCP (d) EMKC (e) EMKC-AVQ

Fig. 4. XCP, RCP, EMKC, and EMKC-AVQ in multi-link topology $\mathscr{T}_4$.

### 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. 1b, where it takes XCP over 1.5 min 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 Figs. 1c and 4c, it takes RCP around 30 s (i.e., 15 RTTs) to reach the stationarity.

For EMKC and small $N\alpha \ll C$, [33] shows that flows reach fairness in $\Theta(C\log N/(N\alpha))$ steps, which scales *linearly* with resource capacity $C$. In Fig. 1c, for instance, it takes two EMKC flows over 4 min 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 $\alpha$, EMKC's linear rate of convergence is clearly undesirable, especially in high-speed networks. To achieve capacity-independent convergence, $\alpha$ 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 $C$ [33]. In general, there is no algorithmic way for end-flows to select their $\alpha$ 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 $\alpha$ keeps the protocol from quickly converging to fairness.

### 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 10 000 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 loss scales linearly with the number of connections and becomes worse if one increases $\alpha$ 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 $\alpha$ to 2 mb/s. As Fig. 5a shows, the instantaneous rate reaches 13 mb/s and the transient overshoot lasts for over 50 s. Moreover, this situation becomes even worse when the number of competing flows increases. As seen in Fig. 5b, 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 $\alpha$. 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 because 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

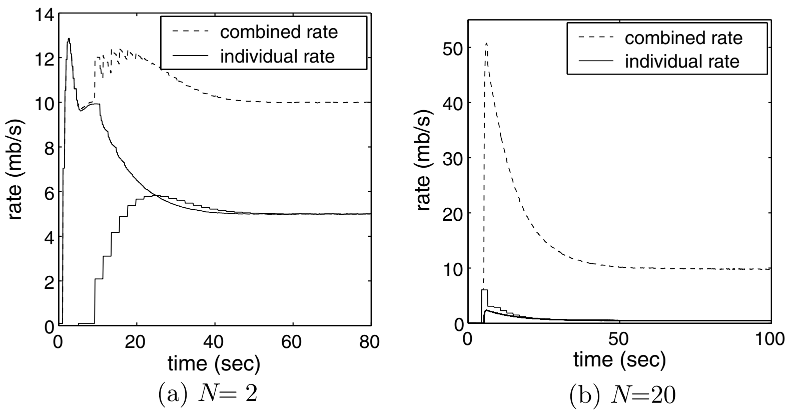


Fig. 5. Transient overshoot of EMKC-AVQ ($\alpha = 2$ mb/s, $\beta = 0.5$, and $\tau = 0.2$).

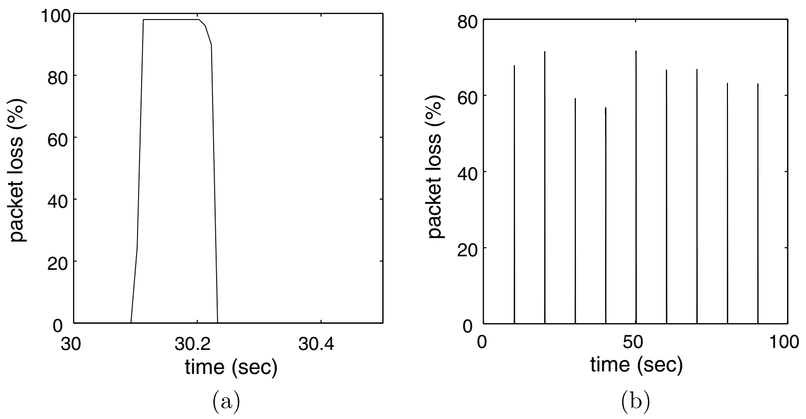


Fig. 6. Transient overshoot of RCP ($\alpha = 0.4$ and $\beta = 1$).

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 $\mathscr{T}_5$. One flow starts first and the other fifty flows join after 30 s. 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. 6a, 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. 6b, in which every 10 s ten RCP flows arrive at a 100-mb/s link and each flow has a random lifetime between 1 and 15 s. 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.

## 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.

[5] Note that RCP does not specify how flows react to packet loss or recover dropped packets ([7] uses very large buffers for all simulations to prevent packet loss). However, a common technique [16] is to use TCP's recovery mechanism (i.e., reducing rate in half) until all lost packets are recovered.

### 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 [6] 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 1** [2]. 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 [16,33], 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) [33]; 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 rewrite the general form of MKC [33] as follows:

$$x_r(n) = (1 - \beta p_r(n - D^{\leftarrow}_{r,b_r}))x_r(n - D_r) + \alpha, \tag{6}$$

where

$$p_r(n) = p\left(\sum_{s=1}^{N} R_{sb_r} x_s(n - D^{\rightarrow}_{s,b_r})\right). \tag{7}$$

Notice that the sum in (7) 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 (6), (7) 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 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 [33] does not hold.

Analysis below uses notation $x_s \rightarrow x_r$ 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. 7a and max–min

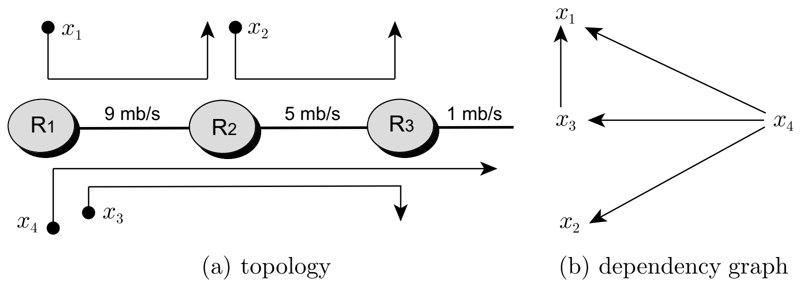


Fig. 7. Example that shows the effect of unresponsive flows.

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. 7b.

**Lemma 1.** *For any system with max–min feedback that can stabilize its bottleneck assignment $b_1, \ldots, b_N$, the resulting dependency graph of (6), (7) 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 $k \geqslant 2$. 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 $p^*_{i_1} > p^*_{i_2} > \cdots > p^*_{i_k} > p^*_{i_1}$. 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 2.** *Systems (6) and (7) 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 1 and 2, we next prove MKC's stability under any time-invariant bottleneck assignment.

**Theorem 1.** *Under any bottleneck assignment $B$ that does not change over time, MKC (6) and (7) is locally asymptotically stable regardless of delay if and only if for each link $l$, the subsystem composed of link $l$ and flows $\{x_r \mid B_{rl} = 1\}$ is stable regardless of delay.*

**Proof.** Since bottlenecks do not shift and MKC relies on max–min feedback, Lemma 1 implies that the dependency graph is acyclic and bottleneck assignment is consistent. Using Lemma 2, 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 its 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 $A = \mathrm{diag}(A_1, \ldots, A_M)$, 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 1). This is schematically shown in Fig. 8a, 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 $x_{i_1} \to \cdots \to x_{i_k} \to x_{i_1}$ 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 some time

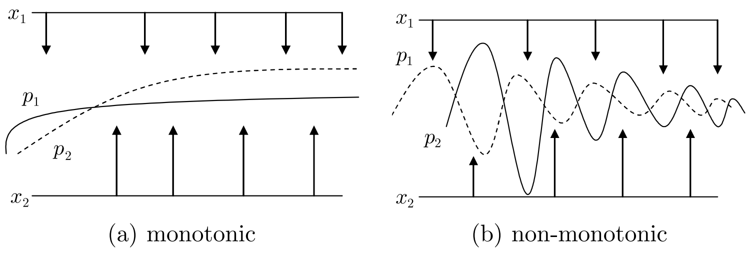


Fig. 8. Types of multi-router feedback.

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. 8b, 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 1.** *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 $0 < \gamma_l \leqslant 1$ 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 $\Delta_l$ time units) calculates and feeds back to the senders the fair rate $g_l(n)$:

$$g_l(n) = \frac{\gamma_l C_l - u_l(n)}{N_l(n)}, \tag{8}$$

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

$$x_r(n) = (1-\tau)x_r(n-D_r) + \tau g_l(n - D_r^{\leftarrow}), \tag{9}$$

where constant $\tau > 0$. Clearly, $x_r(n)$ is in fact an exponential weighted moving average of $g_l(n)$ with weight $\tau$. An alternative interpretation of (9) 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, Eq. (9) 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. [3] 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.,

$$p_l(n) = \frac{y_l(n) - \gamma_l C_l}{y_l(n)}. \tag{10}$$

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 [5] 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 3.** *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 Eq. (9) immediately yields $x_r^* = (\gamma_l C_l - u_l^*)/N_l^*$.

We next show that, under any consistent bottleneck assignment, stability analysis of systems (8) and (9) can be reduced to that of EMKC.

**Theorem 2.** *Under any consistent bottleneck assignment, JetMax (8), (9) 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 [33], we obtain that single-link JetMax is stable for all types of directional and time-varying delay under the same condition on $\tau$. Finally, invoking Theorem 1, 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 paper, this proposed method is very accurate in both `ns2` simulations and Linux experiments.

To better understand Theorem 2, 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 2, 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. 9a. At the same time, as Fig. 9b 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 3, 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

$$b_r = \arg \min_{l \in r : p_l^* = 0} g_l(n). \tag{11}$$

To maintain stability, switching based on the largest packet loss (10) may be performed at any time $n$;

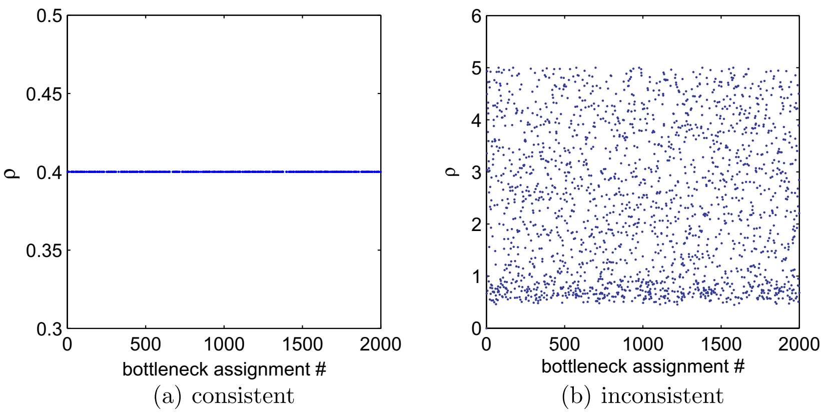


Fig. 9. Spectral radius $\rho(A)$ of systems (8) and (9) with $\tau = 0.6$ under 2000 random bottleneck assignments.

however, that based on (11) is conducted only when flow $r$'s sending rate reaches the $\varepsilon$-neighborhood of stationarity under the current bottleneck assignment. We next prove max–min fairness of the resulting system.

**Theorem 3.** *The stationary resource allocation of JetMax* (9), (11) *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 [2, 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 3 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 1, flow $r$ is not bottlenecked by this link if and only if there exists a flow $s$ accessing $l$ such that

$$x_r^* < x_s^*. \tag{12}$$

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 (12) into

$$\frac{\gamma_l C_l - u_l^*}{N_l^*} < \frac{\gamma_k C_k - u_k^*}{N_k^*}. \tag{13}$$

According to (11), 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 2.** 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

$$\forall n \geqslant n_e : \frac{y(n)}{\gamma C} \geqslant 1 - \varepsilon. \tag{14}$$

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

$$\forall n \geqslant n_f : \frac{|x_r(n) - x_r^*|}{x_r^*} \leqslant \varepsilon \quad \forall r. \tag{15}$$

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

**Theorem 4.** *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 (9) 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

$$y(n) = (1-\tau)^{n/D}(y(0) - \gamma C) + \gamma C, \tag{16}$$

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 (14) and writing $n_e$ in terms of RTT steps, we get $|1-\tau|^{n_e} \leqslant \varepsilon$, which yields

$$n_e = \lceil \log_{|1-\tau|} \varepsilon \rceil. \tag{17}$$

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 (9) becomes

$$x_r(n) = (1-\tau)x_r(n-D_r) - \tau x_r(n-D_r). \tag{18}$$

Solving this recurrence, we get

$$x_r(n) = (1-\tau)^{n/D_r}(x(0) - x_r^*) + x_r^*, \tag{19}$$

which shrinks to $(1-\varepsilon)$-fairness in $n_f = \lceil \log_{|1-\tau|} \varepsilon \rceil$ RTT steps following the technique we used to obtain (17). □

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 in the paper). Also observe from (16) and (19) that $0 < \tau < 1$ is required to guarantee monotonicity of the controller. Thus, all JetMax experiments in this paper 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 [1,11,28]. Specifically, as suggested in [1,11], $N_l(n)$ 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 [28] 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.

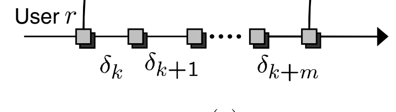
Our solution to this problem is based on the following observations. For a given flow $r$, assume that $\delta_k$ is the inter-packet departure delay between packets $k$ and $k+1$ at the source and $\delta'_k$ is the corresponding inter-packet arrival delay at link $l$. Fig. 10a illustrates this notation and shows that the router's control interval $\Delta_l$ 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

$$\sum_{i=k+1}^{k+m-1} \delta'_i \leqslant \Delta_l \leqslant \sum_{i=k}^{k+m} \delta'_i, \tag{20}$$

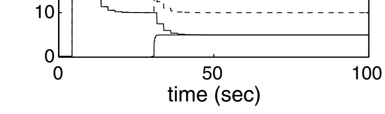
where $k+m$ is the packet that arrives immediately after the end of this interval. This further yields

$$\lim_{\Delta_l \to \infty} \frac{\sum_{i=k}^{k+m} \delta'_i}{\Delta_l} = \lim_{\Delta_l \to \infty} \frac{\sum_{i=k+1}^{k+m-1} \delta'_i}{\Delta_l} = 1. \tag{21}$$

Generalizing this relation to all $N_l$ flows bottlenecked by $l$ and taking the summation of inter-packet delays over all such flows, we have



(a)



(b)

Fig. 10. (a) The relationship between control interval $\Delta_l$ and inter-packet interval $\delta_k$; (b) JetMax ($\tau = 0.6$ and $\gamma = 1$) with the naive bottleneck-switching scheme in $\mathcal{T}_1$.

$$\lim_{\Delta_l\to\infty} \frac{\sum_{r=1}^{N_l}\sum_{i=k}^{k+m}\delta'_i}{\Delta_l} = N_l. \quad (22)$$

Even though in general $\delta_k$ does not equal to $\delta'_k$, 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 (22), leads to

$$\lim_{\Delta_l\to\infty} \frac{\sum_{r=1}^{N_l}\sum_{i=k}^{k+m}\delta_i}{\Delta_l} = N_l. \quad (23)$$

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 $\Delta_l$. From (23), we have that the value $\widetilde{N}_l = \sum_{r=1}^{N_l}\sum_{i=0}^{m}\delta_{k+i}/\Delta_l$ converges to the true number of flows $N_l$ as $\Delta_l$ 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 $\Delta_l$.

We finally remark that choosing a large interval $\Delta_l$ 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 $\Delta_l$ (say, 100 ms). Thus, throughout the paper 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.

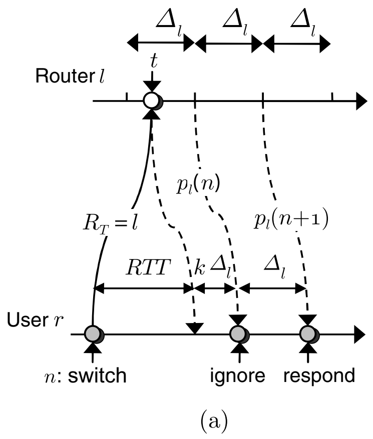
Upon each packet arrival, link $l$ increments $R_C$ by one and then examines its local packet loss $p_l(n)$ and the one carried in the packet. If both packet-loss values are zero, the router checks if its local average rate $g_l(n)$ is less than the one carried in the header. If either case is true, 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 $\Delta_l$, 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 $\Delta_l$, the router obtains the responsive and unresponsive rates by normalizing $w_l(n)$ and $u_l(n)$ by $\Delta_l$. 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 $\mathscr{T}_1$, 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 s, and experience round-trip delay 2020 and 220 ms, respectively. The simulation result is plotted in Fig. 10b, in which $x_2$ initially overflows the link's capacity by 500% and then maintains non-zero packet loss for over 15 s. As we discuss below, this phenomenon arises as the result of improper management of bottleneck switching.

For the illustration in Fig. 11a 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 $\Delta_l$. Notice that flow $r$ is counted



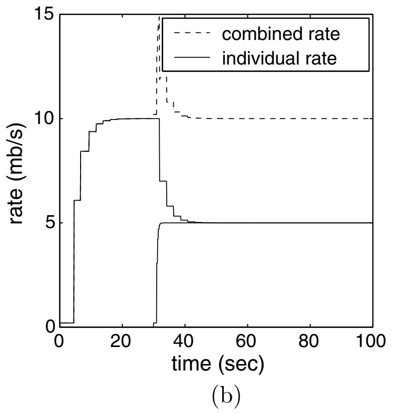


Fig. 11. (a) The scenario where the bottleneck switching occurs in the middle of the router's control interval; (b) JetMax ($\tau = 0.6$ and $\gamma = 1$) with the proper bottleneck-switching scheme in $\mathcal{T}_1$.

as *unresponsive* prior to time $t$ and *responsive* after that. This inconsistent inference of membership results in 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 $\Delta_l$ 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. 11a. We can also see from the figure that, using this mechanism, the end-user delays its response to ACKs by one RTT and $1 + k$ (where $0 \leqslant k \leqslant 1$) $\Delta_l$ after each switching. Simulation of the resulting JetMax is illustrated in Fig. 11b, in which the initial "spike" present in Fig. 10b 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. 11b 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

$$s_k^+ = s_k \frac{x_r^+(n)}{x_r(n)}. \tag{24}$$

For each incoming packet during interval $\Delta_l$, 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 $\Delta_l$ 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 than $\gamma_l C_l$ and decline it otherwise. Note that when computing $g_l(n)$ in (8) and $p_l(n)$ in (10) 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 $y_l^+(n)$:

$$p_l^+(n) = 1 - \frac{\gamma_l C_l}{y_l^+(n)},$$
$$g_l^+(n) = \frac{\gamma_l C_l - u_l^+(n)}{N_l}. \tag{25}$$

Clearly, no extra latency is introduced by this mechanism and each approved rate adjustment takes exactly one RTT (instead of two RTTs if (8) and (10) were based on actual rates). The result of this implementation is shown in Fig. 12a, in which the system never overflows the link and converges to fairness monotonically.

### 6.5. Calculating reference rate

Intuitively, when applying control equation (9), 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 $\mathrm{RTT} + (1 + k)\Delta_l$ where $0 \leqslant k \leqslant 1$.

### 6.6. Packet format

The header format of a JetMax packet is illustrated in Fig. 12b. 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 $\delta_k$ 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 [16].

## 7. Simulations

### 7.1. Behavior in $\mathscr{T}_2$, $\mathscr{T}_3$, $\mathscr{T}_4$, and $\mathscr{T}_5$

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 paper is available in [13].

Performance of JetMax in $\mathscr{T}_2, \mathscr{T}_3, \mathscr{T}_4$, and $\mathscr{T}_5$ is shown in Fig. 13, in which the protocol demonstrates

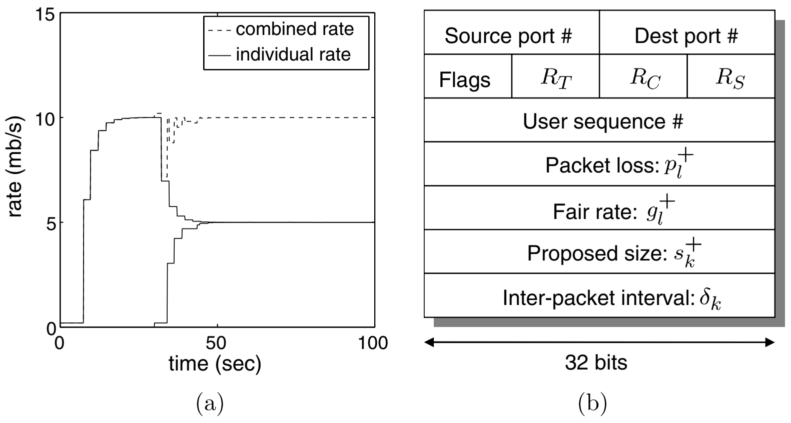


Fig. 12. (a) JetMax ($\tau = 0.6$ and $\gamma = 1$) with proposed rate in $\mathscr{T}_1$; (b) format of the JetMax packet header.

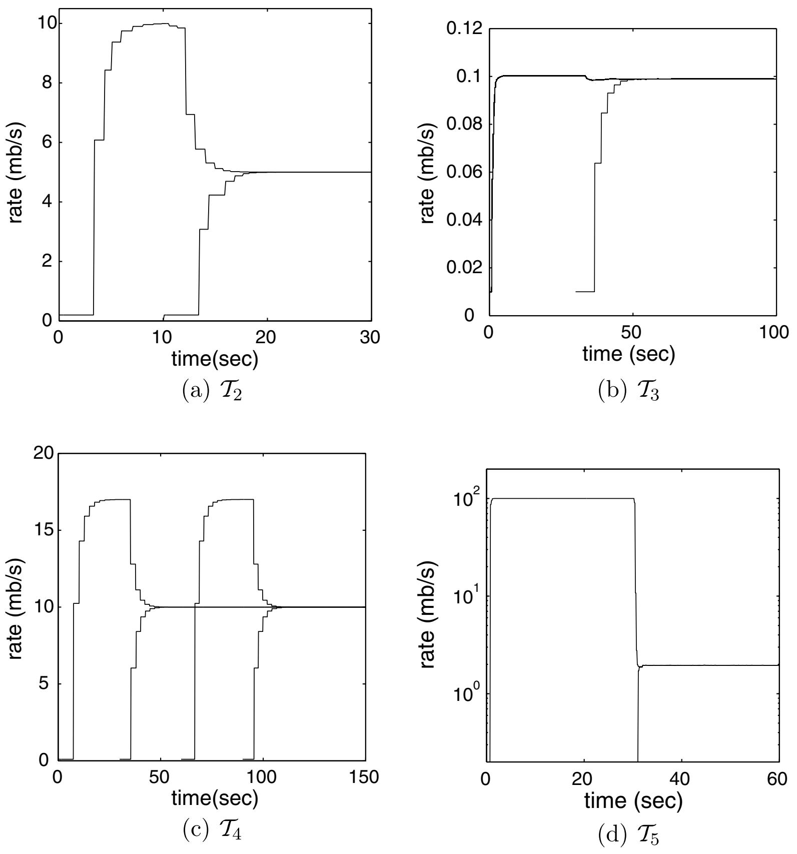


Fig. 13. Performance of JetMax ($\tau = 0.6$ and $\gamma = 1$) in ns2.

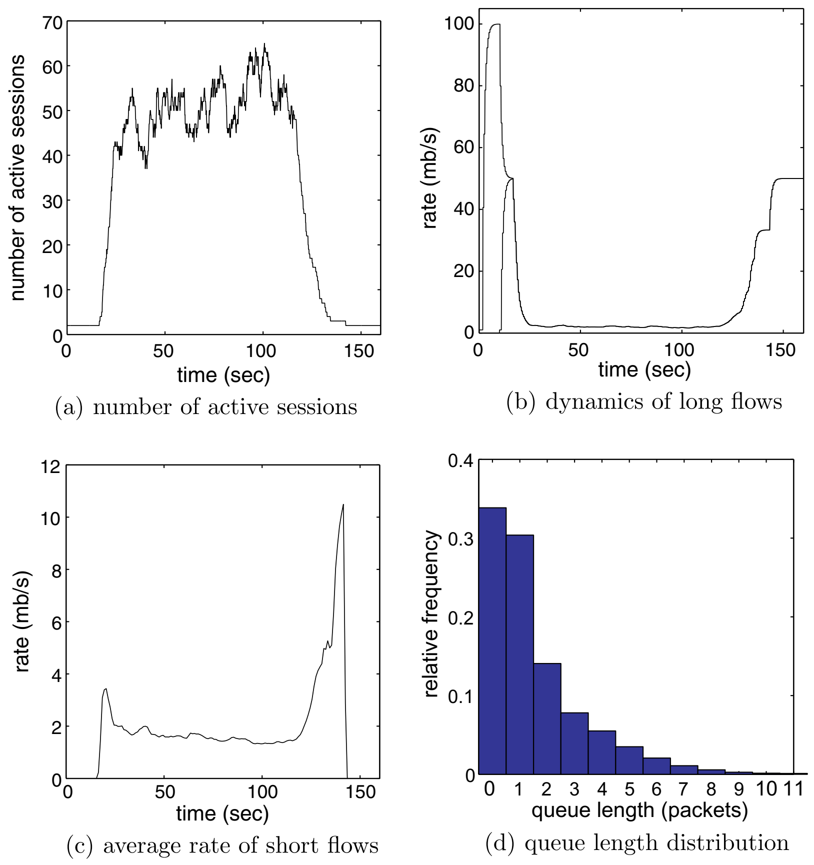
monotonic convergence, max–min allocation of bottleneck resource in the equilibrium, effective handling of bottleneck selection, and loss-free operation in both the transient phase and steady state. Numerical data from the simulations also show that the system never overshoots the link's capacity or 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. 13a when both flows start consume three RTTs (i.e., 2.6 s) and are necessary for the flows to deal with initial router assignment and bottleneck selection.

## 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 [10]. 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 two 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 $\lambda = 0.2$ s and the duration of each flow is drawn from a log-normal distribution [24] with mean $\omega = 10$ s. 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. 14a. 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.

As seen in Fig. 14b, one long flow starts first and quickly reaches link utilization. After the second long flow joins 5 s 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



(a) number of active sessions (b) dynamics of long flows (c) average rate of short flows (d) queue length distribution

Fig. 14. Single-link performance of JetMax ($\tau = 0.6$ and $\gamma = 1$) in the presence of mice flows.

(i.e., 50 mb/s). At time 15 s, mice flows start joining and leaving the network. Since on average there are 50 short and two long flows in the system, the *expected* fair rate is $100/52 = 1.92$ mb/s per flow. This prediction is confirmed in Fig. 14b, where the sending rates of the long flows remain within $[1.7, 2.0]$ mb/s during the period between $[30, 120]$ s. 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. 14c 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 s, 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. 14d, 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, we set $\Delta_l$ to be uniformly random in $[100, 300]$ ms to test JetMax's performance when routers have heterogeneous control intervals. As shown in Fig. 15b, the long flow starts first and converges to the capacity of $R_2$. Short flows accessing $R_1$ start joining the system after 15 s. 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

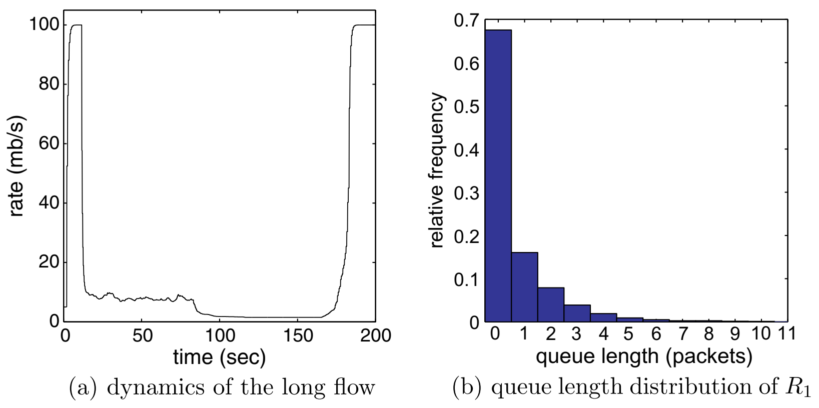


Fig. 15. Multi-link performance of a long JetMax flow ($\tau = 0.6$ and $\gamma = 1$) in the presence of mice traffic.

neighborhood of the average fair rate 400/52 = 7.7 mb/s. At time 80 s, 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. 15b 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 $\Delta_l$ 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 $\mathscr{T}_1$ with 90% packet loss in its return path was almost identical to that in the loss-free environment previously shown in Fig. 12a. 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. 16a, where we use $\mathscr{T}_1$ 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 3, it is not difficult to obtain that

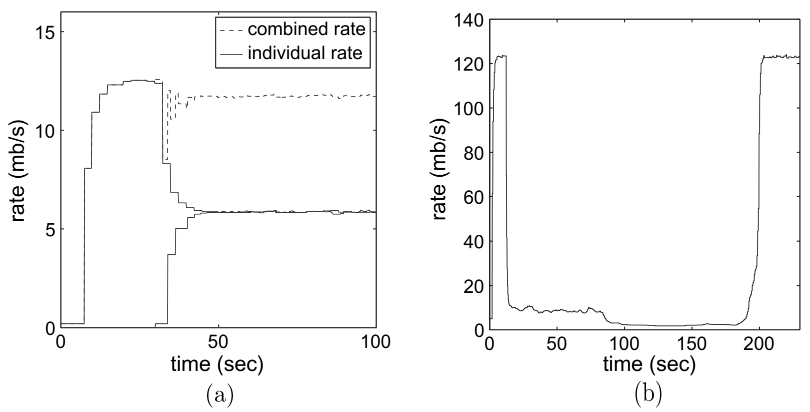


Fig. 16. JetMax ($\tau = 0.6$ and $\gamma = 1$) under random packet loss: (a) $\mathscr{T}_1$ with 10% forward-path loss; (b) "parking lot" topology with mice flows and random loss.

$$x_r^* = \frac{\gamma_l C_l - u_l^*}{\alpha_l N_l}, \tag{26}$$

where $\alpha_l$ is given by

$$\alpha_l = \frac{\sum_{r \in S_l} \alpha_{r,l}}{N_l}, \tag{27}$$

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 (26) and (27), 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 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. 15b with 500 mice flows per link, 10% random loss on each link in the forward path, and 50% loss in the backward path. Fig. 16b 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 $\gamma$ to 1. We also neglect the first 20 s of the simulations to avoid transient phase effects and bottleneck switching, and compute efficiency statistics by averaging 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. 17a 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. 17b, 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 $\tau$ to 0.6, round-trip delay $D$ of all flows to 220 ms, and control interval $\Delta_l$ of the bottleneck router to 100 ms. Notice that as discussed in Section 6.5, proper calculation of the reference rate takes $RTT + (1+k)\Delta_l$ time units, where $k \in [0,1]$. According to Theorem 4, 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] s. This prediction is confirmed by simulation results illustrated in Fig. 18, where end-users spend around 2.4 s before reaching both efficiency and fairness over a wide range of link bandwidths.

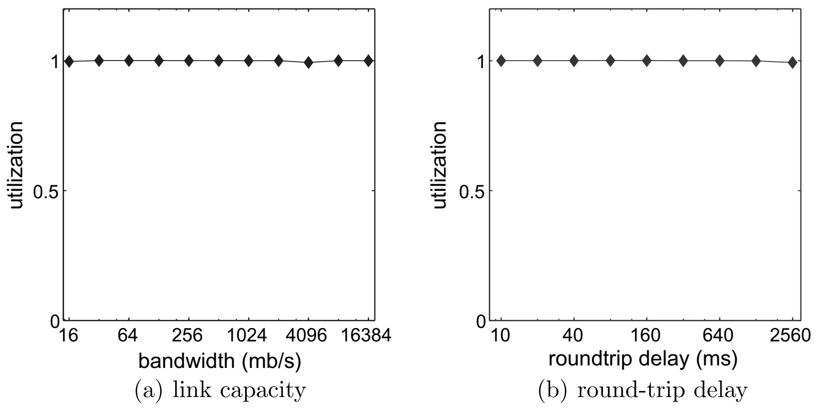


Fig. 17. Utilization of the bottleneck in JetMax ($\tau = 0.6$ and $\gamma = 1$) under different link capacities and round-trip delays in ns2.

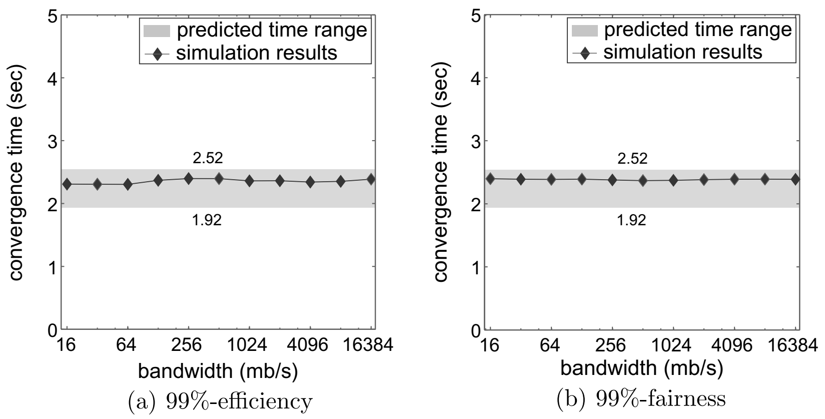


Fig. 18. Convergence time of JetMax ($\tau = 0.6$ and $\gamma = 1$) as a function of bottleneck capacity $C$ in ns2.

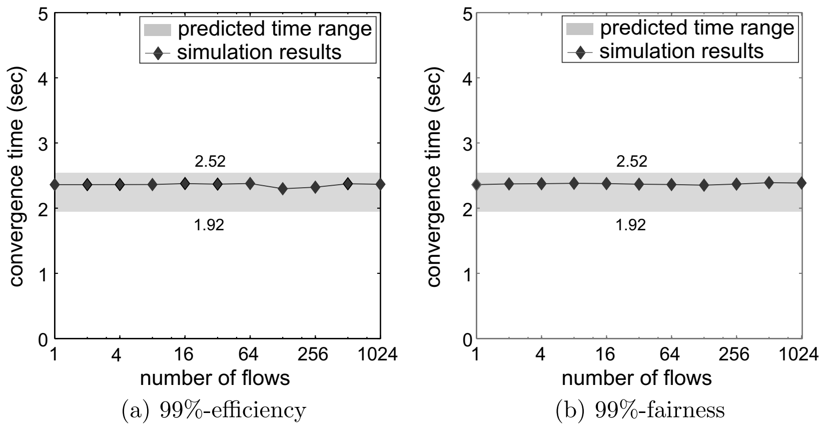


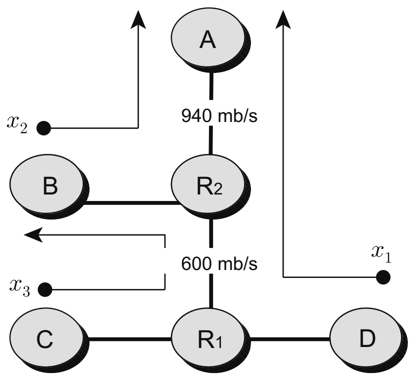
Fig. 19. Convergence time of JetMax ($\tau = 0.6$ and $\gamma = 1$) as a function of the number of users $n$ in ns2.

Additionally, Theorem 4 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. 19, under different numbers of flows (from 1 to 1024), it takes JetMax the same six steps to enter the 1%-neighborhood of both efficiency and fairness.

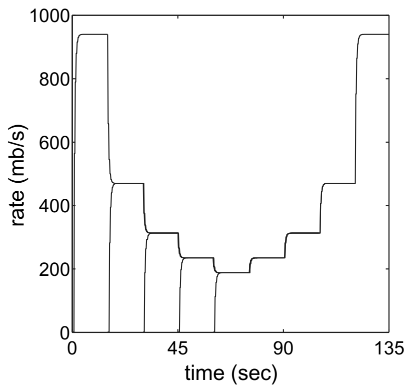
## 8. Linux performance

We finish the paper by examining performance and implementation overhead of JetMax in Linux software routers. The main goal of this study is to advance beyond 10 mb/s cases studied in the literature [32] and achieve true gigabit speeds where AQM algorithms would have the most impact in practice. For the experiments reported in this paper, we use two Linux routers shown in Fig. 20a, 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 $\gamma_l$.

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 Linux kernels do in fact support floating-point operations (despite a popular



(a) topology



(b) single-router test

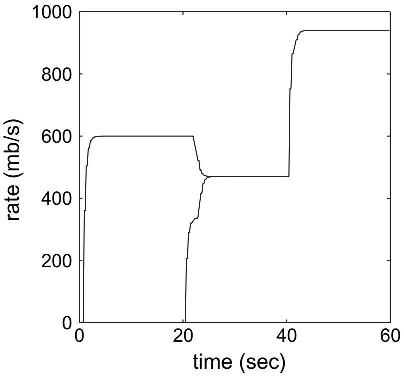
Fig. 20. Single-router Linux experiments with JetMax ($\tau = 0.6$).

belief to the contrary [32]) 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 $\Delta$-interval timing.

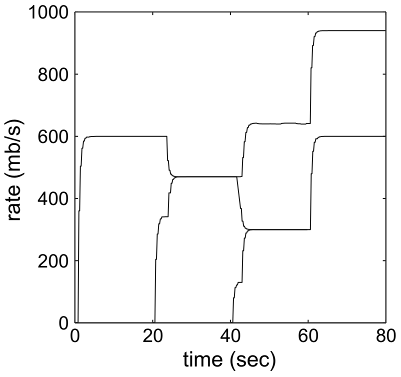
For the first test, we run five flows from host $B$ to $A$ in Fig. 20a to examine the ability of JetMax to utilize high-bandwidth links and support multiple senders/receivers per end-host. Each flow starts with a 15-s delay and lasts for 75 s. The performance of JetMax for this setup is shown in Fig. 20b. Notice in the figure that the first flow converges to 99% of 940 mb/s in 1.3 s and maintains its steady-state rate without oscillations. As subsequent flows arrive, they take 1.2 s (which is six 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 s to 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. 20a 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. 21a, 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 $t = 40$ s. 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. 21b, 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



(a) two-flow test



(b) three-flow test

Fig. 21. Dual-router Linux experiments with JetMax ($\tau = 0.6$).

(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. Conclusion

This paper 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 paper 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.

## References

[1] J. Aweya, M. Ouellette, D.Y. Dontuno, A. Simple, Scalable and provably stable explicit rate computation scheme for flow control in communication networks, Int. J. Commun. Syst. 14 (6) (2001) 593–618.

[2] D. Bertsekas, R. Gallager, Data Networks, Prentice-Hall, 1992.

[3] F. Blanchini, R.L. Cigno, R. Tempo, Robust rate control for integrated services packet networks, IEEE/ACM Trans. Netw. 10 (5) (2002) 644–652.

[4] M. Christiansen, K. Jeffay, D. Ott, F.D. Smith, Tuning RED for Web traffic, in: Proceedings of the ACM SIGCOMM, August 2000, pp. 139–150.

[5] S. Deb, R. Srikant, Rate-based versus queue-based models of congestion control, in: Proceedings of the ACM SIGMETRICS, June 2004, pp. 246–257.

[6] R. DeCarlo, M.S. Branicky, S. Pettersson, B. Lennartson, Perspectives and results on the stability and stabilizability of hybrid systems, Proc. IEEE 88 (7) (2000) 1069–1082.

[7] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, N. McKeown, Processor sharing flows in the internet, in: Proceedings of the IEEE IWQoS, June 2005.

[8] A. Falk, D. Katabi, Specification for the Explicit Control Protocol (XCP), USC/ISI, Technical Report, October 2005.

[9] S. Floyd, High-speed TCP for Large Congestion Windows, IETF RFC 3649, December 2003.

[10] S. Floyd, E. Kohler, Internet research needs better models, ACM SIGCOMM Comput. Commun. Rev. 33 (1) (2003) 29–34.

[11] C. Fulton, S.-Q. Li, C.S. Lim, An ABR feedback control scheme with tracking, in: Proceedings of the IEEE INFOCOM, April 1997, pp. 805–814.

[12] C.V. Hollot, V. Misra, D. Towsley, W.-B. Gong, On designing improved controllers for AQM routers supporting TCP flows, in: Proceedings of the IEEE INFOCOM, April 2001, pp. 1726–1734.

[13] JetMax@TAMU. [Online]. <http://irl.cs.tamu.edu/projects/mkc/>.

[14] C. Jin, D. Wei, S.H. Low, FAST TCP: motivation, architecture, algorithms, performance, in: Proceedings of the IEEE INFOCOM, March 2004, pp. 2490–2501.

[15] R. Johari, D.K.H. Tan, End-to-end congestion control for the internet: delays and stability, IEEE/ACM Trans. Netw. 9 (6) (2001) 818–832.

[16] D. Katabi, M. Handley, C. Rohrs, Congestion control for high bandwidth delay product networks, in: Proceedings of the ACM SIGCOMM, August 2002, pp. 89–102.

[17] F.P. Kelly, A.K. Maulloo, D.K.H. Tan, Rate control for communication networks: shadow prices, proportional fairness and stability, J. Oper. Res. Soc. 49 (3) (1998) 237–252.

[18] S. Kunniyur, AntiECN marking: a marking scheme for high bandwidth delay connections, in: Proceedings of the IEEE ICC, May 2003, pp. 647–651.

[19] S. Kunniyur, R. Srikant, Stable, scalable, fair congestion control and AQM schemes that achieve high utilization in the internet, IEEE Trans. Automat. Contr. 48 (11) (2003) 2024–2029.

[20] S. Kunniyur, R. Srikant, Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management, in: Proceedings ACM SIGCOMM, August 2001, pp. 123–134.

[21] S. Liu, T. Basar, R. Srikant, Pitfalls in the fluid modeling of RTT variations in window-based congestion control, in: Proceedings of the IEEE INFOCOM, March 2005, pp. 1002–1012.

[22] S.H. Low, L.L.H. Andrew, B.P. Wydrowski, Understanding XCP: equilibrium and fairness, in: Proceedings of the IEEE INFOCOM, March 2005, pp. 1025–1036.

[23] L. Massoulié, Stability of distributed congestion control with heterogeneous feedback delays, IEEE Trans. Automat. Contr. 47 (6) (2002) 895–902.

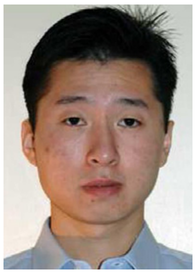
[24] V. Paxson, Empirically derived analytic models of wide-area TCP connections, IEEE/ACM Trans. Netw. 2 (4) (1994) 316–328.

[25] R.S. Prasad, M. Jain, C. Dovrolis, On the effectiveness of delay-based congestion avoidance, in: Proceedings of the PFLDnet, February 2004.

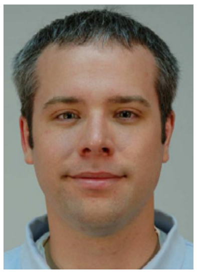
[26] K.K. Ramakrishnan, S. Floyd, D. Black, The addition of explicit congestion notification (ECN) to IP, IETF RFC 3168, September 2001.

[27] J. Wang, D.X. Wei, S.H. Low, Modeling and stability of FAST TCP, in: Proceedings of the IEEE INFOCOM, March 2005, pp. 938–948.

[28] M.K. Wong, F. Bonomi, A novel explicit rate congestion control algorithm, in: Proceedings of the IEEE GLOBECOM, November 1998, pp. 8–12.
[29] B.P. Wydrowski, M. Zukerman, MaxNet: a congestion control architecture for maxmin fairness, IEEE Commun. Lett. vol. 6 (11) (2002) 588–599.
[30] XCP@ISI. [Online]. <http://www.isi.edu/isi-xcp/>.
[31] C.-C. Yu, Autotuning of PID Controllers: A Relay Feedback Approach, Springer-Verlag, 2006.
[32] Y. Zhang, T. Henderson, An implementation and experimental study of the explicit control protocol (XCP), in: Proceedings of the IEEE INFOCOM, March 2005, pp. 1037–1048.
[33] Y. Zhang, S.-R. Kang, D. Loguinov, Delayed stability and performance of distributed congestion control, in: Proceedings of the ACM SIGCOMM, August 2004, pp. 307–318.



**Yueping Zhang** received the B.S. degree in computer science from Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001. He is currently working toward the Ph.D. degree in computer engineering at Texas A&M University, College Station, USA. His research interests include congestion control, delayed stability analysis, active queue management (AQM), router buffer sizing, and peer-to-peer networks.



**Derek Leonard** received the B.A. degree (with distinction) in computer science and mathematics from Hendrix College, Conway, Arkansas, in 2002. Since 2002, he has been a Ph.D. student in the Department of Computer Science at Texas A&M University, College Station, TX. His research interests include peer-to-peer networks, optimization-based graph construction, and large-scale measurement studies of the Internet.

**Dmitri Loguinov** received the B.S. degree (with honors) in computer science from Moscow State University, Russia, in 1995 and the Ph.D. degree in computer science from the City University of New York, New York, in 2002.

Between 2002 and 2007, he was an Assistant Professor in the Department of Computer Science at Texas A&M University, College Station. He is currently a tenured Associate Professor and Director of the Internet Research Lab (IRL) in the same department. His research interests include peer-to-peer networks, video streaming, congestion control, Internet measurement and modeling.