JetMax: Scalable Max-Min Congestion Control for High-Speed Heterogeneous Networks

Yueping Zhang, Derek Leonard, and Dmitri Loguinov*
Texas A&M University, College Station, TX 77843
{yueping, dleonard, dmitri}@cs.tamu.edu

***Abstract*— Recent surge of interest towards congestion control that relies on *single-router* feedback (e.g., XCP [12], RCP [1], [5], MaxNet [24], EMKC [28], VCP [26]) suggests that such systems may offer certain benefits over traditional models of additive packet loss [13]. Besides topology-independent stability and faster convergence to efficiency/fairness [24], it was recently shown [28] that any stable single-router 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 [28] 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 6 control intervals. We finish the paper by comparing JetMax's performance to that of existing methods in `ns2` simulations and discussing its Linux implementation.**

***Index Terms*— Congestion Control, Multi-link Stability, Max-Min Fairness, High-Speed Networks.**

I. Introduction

In the light of TCP's scalability issues in high-speed networks [7], explicit-feedback congestion control has gained renewed interest in the last several years [12], [18], [27], [28]. 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 [12], packet loss [28], single-bit congestion indication [9], [14], [22], queuing delay [11], [23], [24], or link prices [13], [15], [19]. This information helps end-flows converge their sending rates to some social optimum and achieve a certain optimization objective.

Unlike some of the largely ineffective AQM aimed at improving the performance of TCP [3], 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 multi-router stability and convergence performance of several recently proposed AQM approaches: eXplicit Control Protocol (XCP) [12], Exponential Max-min Kelly Control (EMKC) [28], and a hybrid method suggested in [28] that combines EMKC with Adaptive Virtual Queue (AVQ) [15], [16]. We find from this analysis that XCP is prone to instability 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 router) 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 router 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 [5], [12], [24], [28]. 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.

*Supported by NSF grants CCR-0306246, ANI-0312461, CNS-0434940, and CNS-0519442.

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

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.
- *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 several 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 II and identify their problems in Section III. We then highlight the importance of studying multi-link stability of max-min systems in Section IV. Following that, we introduce JetMax in Section V and discuss its implementation issues in Section VI. We then demonstrate JetMax's performance through `ns2` simulations and Linux experiments in Sections VII and VIII, respectively. We conclude the paper in Section IX.

II. 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 $\{x_r(t)\}_{r=1}^N$. Further assume that the RTT of each flow is denoted by $D_r(t)$ and the forward/backward delays of user r to/from router l by $D^{\rightarrow}_{r,l}(t)$ and $D^{\leftarrow}_{r,l}(t)$, respectively. The aggregate arrival rate of all users at router 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 routers l used by flow r.

Since its appearance in 2002, XCP [12] has become a defacto standard for explicit congestion control in IP networks [6]. 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, the paper shows that the combined rate $y_l(t)$ is stable if control parameters α and β satisfy $0 < \alpha < \pi/4\sqrt{2}$ and $\beta = \alpha^2\sqrt{2}$.

XCP's design goals [12] include max-min fairness and high link utilization; however, a recent study of its equilibrium properties [18] shows that XCP does not generally achieve max-min fairness in multi-router 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 [18]. Another study [27] reports experiments with a 10-mb/s XCP Linux router and identifies several implementation issues including uncertainty in accurate selection of C_l, sensitivity to receiver buffer size, and various problems with partial deployment.

The recently proposed Rate Control Protocol (RCP) [5] is a rate-based max-min AQM algorithm in which each router 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 routers if their suggested rate is less than the one currently present in the header. Routers decide the fair rate $r_l(t)$ by implementing a controller

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

where Δ is the router's control interval, α and β are constants, d_l is a moving average of RTTs sampled by router l, C_l is its capacity, and $q_l(t)$ is queue length at time t. Even though the steady-state equations of RCP and XCP are the same [1], RCP has lower implementation overhead, offers quicker transient dynamics, and achieves max-min fair rate allocation [5].

Two additional max-min methods are inspired by Kelly's optimization framework [13] and aim to improve stability and convergence properties of traditional models of additive packet loss [16], [19]. The first approach called MaxNet [24] obtains feedback $f_r(t) = \max_{l\in r} p_l(t)$ from the most congested router 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

$$\dot{p_l}(t) = \frac{y_l(t) - \gamma C_l}{\gamma C_l}, \quad (2)$$

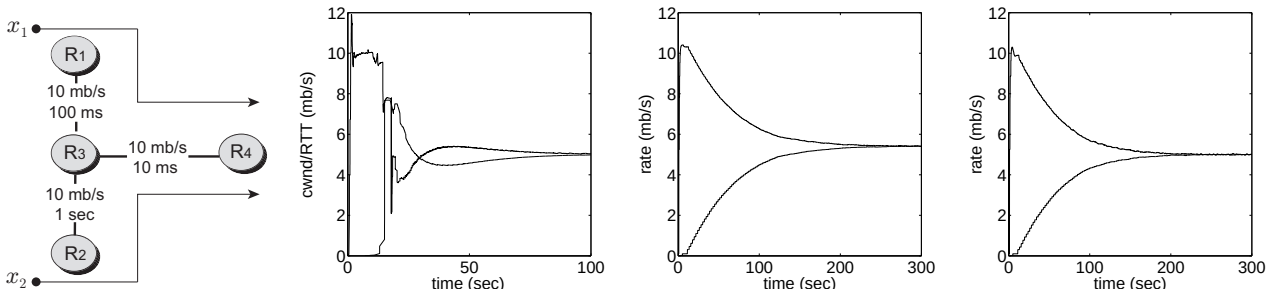
where $0 < \gamma < 1$ is the desired link utilization.

The second method is Exponential Max-min Kelly Control (EMKC) [28], 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

$$x_r(n) = x_r(n - D_r) + \alpha - \beta p_r(n) x_r(n - D_r), \quad (3)$$

where D_r 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 [28]

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



(a) topology $\mathcal{T}_1$ (b) XCP (c) EMKC (d) EMKC-AVQ

Fig. 1. XCP, EMKC, and EMKC-AVQ under constant heterogeneous delay.

For a single-link network, system (3)-(4) is locally asymptotically stable for all time-varying delays satisfying

$$\lim_{n\to\infty} n - [D^{\rightarrow}_{s,l}(n) + D^{\leftarrow}_{r,l}(n)] = \infty, \ \forall r, s, l. \tag{5}$$

Due to the steady-state overshoot of link's capacity [28], EMKC does not reach max-min fairness. However, as suggested in [28], EMKC can be combined with AVQ [15] to guarantee max-min fair rates and zero packet loss in the stationary state.

III. 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 feedback delay, stability in multi-link scenarios, convergence behavior, and overshoot properties in transient and equilibrium states.

A. 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 router, as well as the corresponding backward feedback delay, are dynamic (often random) metrics [21]. Traditional models of congestion control [12], [16], [19], [23] 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 [17]. It thus becomes important to examine how protocols behave in highly heterogeneous environments and whether fluctuating feedback delay may cause them to oscillate.

Besides stability, ideal congestion control should exhibit fast convergence to both efficiency and fairness, avoid overshooting capacity in transient and stationary states, and converge to the desired link utilization γ. While the first few factors are mostly important to end-users, the last metric is of interest to network operators, who usually run their backbones at well below capacity and may not appreciate protocols (such as [11], [12]) 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 in 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.

B. Methodology

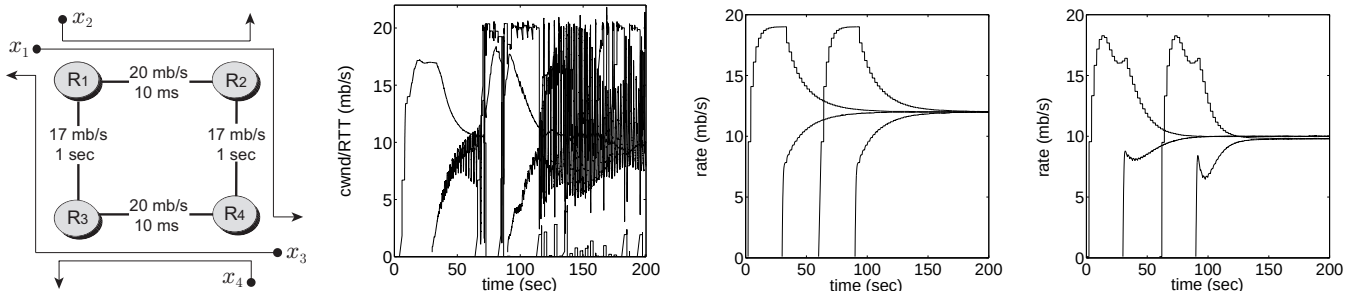
Our main focus in this comparison study is on XCP [12] and EMKC [28] as two completely different approaches to max-min congestion control. At the time of this writing, RCP [5], MaxNet [24], and VCP [26] did not have a publicly available implementation; however, we found that a combination of EMKC and AVQ [15] possessed transient and stationary behavior similar to that of MaxNet. Recall that AVQ dynamically adjusts the virtual capacity of each link until the arrival rate $y_l(t)$ is stabilized at γC_l, where γ is the desired link utilization. This method is similar to the price integrator (2) in MaxNet with the exception that AVQ is not feedback-specific.

Throughout this section, we use `ns2` simulations with XCP and AVQ code that comes with the simulator (version 2.27), and EMKC code used in [28]. We also experimented with the modified XCP code from ISI [25] and found it to offer no stability benefits over the original code. We thus limit our XCP discussion to the algorithms used in [12].

We should finally note 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 provide their exhaustive evaluation under "realistic" Internet conditions.

C. 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. 1(a), where two flows x_1 and x_2 with round-trip delays 220 and 2020 ms, respectively, start with a 5-second delay and share a 10-mb/s link. For XCP, we use the parameters suggested in [12] (i.e., $\alpha = 0.4$ and $\beta = 0.226$) and set the buffer size sufficiently large (i.e., at least $C_l \times RTT$). As Fig. 1(b) shows, XCP is stable under heterogeneous delay, even though it exhibits oscillations and relatively slow (compared to the case of homogeneous D) convergence to fairness.



(a) topology $\mathcal{T}_2$ (b) XCP (c) EMKC (d) EMKC-AVQ

Fig. 2. XCP, EMKC, and EMKC-AVQ in a multi-link scenario.

For EMKC we set $\alpha = 0.2$ mb/s, $\beta = 0.5$, $\Delta = 100$ ms, and repeat the simulation in $\mathcal{T}_1$. The result is plotted in Fig. 1(c), which demonstrates that EMKC converges to the stationary state much more smoothly than XCP; however, it spends over 250 seconds before reaching fairness and eventually overshoots link's capacity by 8%. Although EMKC's convergence rate can be improved by increasing α, this leads to more steady-state packet loss and larger overshoot [28]. We delay further discussion of this issue until later in the section.

The third 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 $y_l(n)$ 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 Δ time units and adjust the virtual capacity $\tilde{C}_l$ at the end of this interval

$$\tilde{C}_l(n) = \tilde{C}_l(n-\Delta) + \frac{\tau\Delta(\gamma C_l - y_l(n))}{D_{max}}, \qquad (6)$$

where $\tau = 0.2$ is the gain parameter used throughout this paper, γ is the desired link utilization, D_{max} is the maximum RTT of end-flows, and C_l is the true capacity of the link.[2] The final step of EMKC-AVQ is to limit $\tilde{C}_l$ to the range $(-\infty, \gamma C]$ 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. 1(d), which indicates that EMKC-AVQ is indeed max-min fair in the steady state (i.e., both flows achieve 5 mb/s) as well as stable under heterogeneous delays; however, the convergence rate to fairness remains painfully slow (i.e., over 200 seconds).

D. Multi-link Stability

Our next stability issue is to examine the performance of these protocols in multi-router networks where bottlenecks shift over time and there exists a possibility for incorrect inference of the most-congested router. For the purpose of this section, we study the four-bottleneck case $\mathcal{T}_2$ shown in Fig. 2(a), where four flows $x_1, \ldots, x_4$ are routed over a grid-type network. We customize the routing rules at nodes R_1 and R_4 to always route their traffic (including any ACKs) in the clockwise direction. This ensures that acknowledgments of flow x_1 travel together with flow x_3 and vice versa. At the same time, the acknowledgments of flows x_2 and x_4 are routed along their corresponding shortest paths (i.e., $R_2 - R_1$ and $R_3 - R_4$). Flows start in sequence from x_1 to x_4 with a 30-second delay. Given this order of user join, the system should evolve through two separate stages, where flows x_1 and x_3 originally converge to 17 mb/s and then shift their bottlenecks to accommodate flows x_2 and x_4. The final max-min assignment of rates is 10 mb/s for each flow.

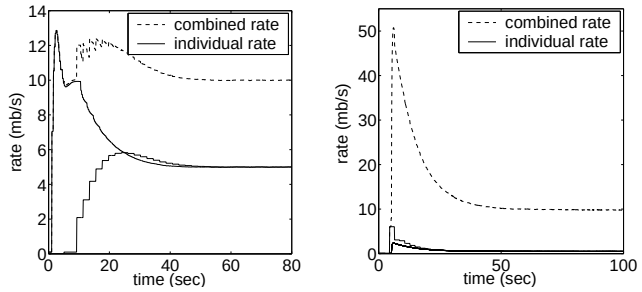
Fig. 2(b) shows the behavior of XCP in $\mathcal{T}_2$. Notice in the figure that the protocol not only oscillates for over 200 seconds, but also denies service to flow x_3, which never obtains its share of the link even in the average sense. The reason for oscillation can be traced to the fact that both x_1 and x_3 continuously switch between their bottlenecks and are unable to settle down in the selection of their most-congested router. This is caused by non-monotonicity of feedback at each router, discontinuous control actions of end-users, and random fluctuation of the RTT that forces XCP to become unstable on small timescales. In contrast, EMKC in Fig. 2(c) and EMKC-AVQ in Fig. 2(d) have no visible stability problems and converge their sending rates exactly as expected.

E. 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. 1(b), where it takes XCP over 1.5 minutes to reach fairness on a 10 mb/s link. At the time of this writing, there are no known expressions for XCP's convergence rate to efficiency or fairness and future analysis of these metrics appears difficult due to the complex behavior of the controller under delay.

For EMKC and small $N\alpha \ll C$, [28] shows that flows reach fairness in $\Theta(C \log N/(N\alpha))$ steps, which scales *linearly* with resource capacity C. In Fig. 1(c), for instance, it takes two EMKC flows over 4 minutes to reach fairness on a 10-mb/s link. Furthermore, the major problem with EMKC's convergence rate to fairness is the tradeoff between convergence speed and stationary packet loss in the network. For small fixed α, EMKC's linear rate of convergence is clearly undesirable, especially in high-speed networks. To achieve capacity-independent convergence, α must be on the order

[2] All delays are computed using XCP's smoothed EWMA estimator with the default weight 0.4 and (6) is normalized by D_{max} to ensure stability of the resulting system under delayed feedback [16].



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

Fig. 3. Transient overshoot of EMKC-AVQ ($\alpha = 2$ mb/s, $\beta = 0.5$ and $\tau = 0.2$).

of C, which results in large stationary packet loss since the amount of steady-state overshoot $N\alpha/\beta$ is now comparable to C [28]. In general, there is no algorithmic way for end-flows to select their α so as to keep loss low and convergence to fairness quick. This is one of the main drawbacks of EMKC.

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

F. Overshoot Properties

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

Among the three controllers in this comparison study, 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 α to accelerate the convergence rate to fairness.

EMKC's problem of steady-state packet loss can be overcome by AVQ; however, the latter may exhibits *transient* overshoot before settling in its max-min fair stationary state. To understand this effect in detail, we repeat the simulation in topology $\mathcal{T}_1$ and increase α to 2 mb/s. As Fig. 3(a) shows, the instantaneous rate reaches 13 mb/s and the transient overshoot lasts for over 50 seconds. Moreover, this situation becomes even worse when the number of competing flows increases. As seen in Fig. 3(b), where 20 EMKC-AVQ users share the same 10-mb/s link in $\mathcal{T}_1$, the transient overshoot reaches 400% and lasts for tens of seconds. This situation is a consequence of the steady-state dynamics inherited from EMKC and the same term $N\alpha/\beta$ responsible for the overshoot, which is a linear function of the number of flows N and parameter α. This leads to a similar tradeoff between packet loss and convergence rate as in EMKC.

IV. 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.

A. 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 switch from one stationary point to another. Traditional switching theory [4] 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.

B. Why Bottleneck Assignment is Important

Under max-min feedback [12], [28], 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* router 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) [28]; however, a similar question arises in other max-min feedback systems.

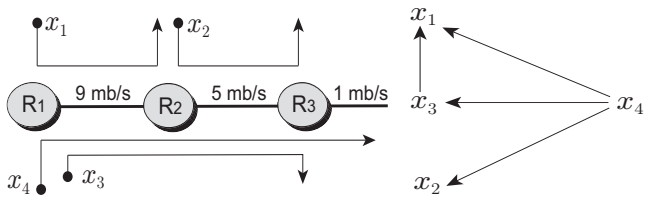
Assume that N users share M routers in the network and suppose that $R \in \mathbb{R}^{N\times M}$ is the routing matrix of end-flows (i.e., $R_{rl} = 1$ if user r uses router l and 0 otherwise). Define b_r to be the bottleneck resource of user r and re-write the general form of MKC [28] as

$$x_r(n) = \big(1 - \beta p_r(n - D^{\leftarrow}_{r,b_r})\big)x_r(n - D_r) + \alpha, \qquad (7)$$

where

$$p_r(n) = p\Big(\sum_{s=1}^{N} R_{sb_r} x_s(n - D^{\rightarrow}_{s,b_r})\Big). \qquad (8)$$

Notice that the sum in (8) 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 router. Even though each flow's feedback in (7)-(8) is still



(a) topology (b) dependency graph

Fig. 4. Example that shows the effect of unresponsive flows.

delayed by only *one* backward delay $D_r^{\leftarrow} = D_{r,b_r}^{\leftarrow}$, each flow s may affect other flows through as many as M forward delays $D_{s,1}^{\rightarrow}, \ldots, D_{s,M}^{\rightarrow}$. 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 [28] 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. 4(a) and max-min assignment of bottlenecks, we have $b_1 = 1, b_2 = b_3 = 2, b_4 = 3$ and the corresponding dependency graph is shown in Fig. 4(b).

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 (7)-(8) is acyclic.

Proof: Suppose that the bottleneck assignment does not change over time and the dependency graph has a directed cycle $x_{i_1} \rightarrow \ldots \rightarrow x_{i_k} \rightarrow x_{i_1}$ for some $k \geq 2$. Notice that since flow x_{i_1} is unresponsive with respect to flow x_{i_2}, its stationary packet loss $p_{i_1}^*$ must be larger than $p_{i_2}^*$ (otherwise, x_{i_1} would have switched its bottleneck to b_{i_2}). Generalizing this to the entire cycle, we immediately get a contradiction $p_{i_1}^* > p_{i_2}^* > \ldots > p_{i_k}^* > p_{i_1}^*$. Assuming a consistent tie-breaking rule obeyed by all flows, the above argument applies to cases where multiple routers 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: System (7)-(8) 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 router 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 that does not change over time, MKC (7)-(8) is locally asymptotically stable regardless of delay if and only if the individual bottlenecks are.

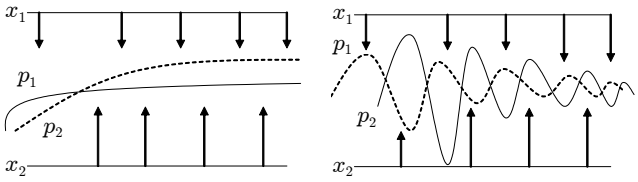
Proof: Since bottlenecks do not shift and MKC relies on max-min feedback, Lemma 1 implies that the dependency graph is acyclic and bottleneck assignment is consistent. Using Lemma 2, there exists at least one router l_1 with no unresponsive flows. Then, it follows that all flows passing through l_1 are bottlenecked by l_1 and their stability is independent of the dynamics of the remaining flows. After the users bottlenecked by l_1 converge to their stationary rates, we can remove l_1 and all of its (constant-rate) flows from the system. The new network still exhibits max-min bottleneck assignment and thus contains some router l_2 that has no unresponsive flows. Repeating this argument for all routers $l_1, \ldots, l_M$, we obtain that the local dynamics of the entire system can be viewed as a system of linear block-diagonal equations with matrix $A = \text{diag}(A_1, \ldots, A_M)$, where $A_l \in \mathbb{R}^{N_l \times N_l}$ is the Jacobian matrix of N_l flows bottlenecked at router l ($\sum_{l=1}^{M} N_l = N$). We conclude 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 routers and avoid dependency cycles, the dynamics of multi-link systems are in fact described by those of individual routers. 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. 5(a), where two flows x_1 and x_2 sample monotonic feedback p_1 and p_2 from two routers 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 $x_{i_1} \rightarrow \ldots \rightarrow x_{i_k} \rightarrow x_{i_1}$ that persists over time. This can be shown using the example of two flows. Suppose cycle $x_1 \rightarrow x_2 \rightarrow x_1$ 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 router b_2. Since the two flows sample packet loss 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 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. 5(b), where both p_1 and p_2 are individually (i.e., without the max function) stable, but create a cyclic dependency graph with potential for instability.

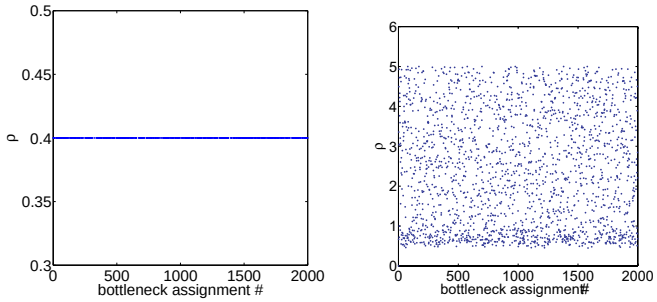
As the XCP examples show, non-monotonic feedback allows flows to continuously switch between bottlenecks and maintain persistent cycles in the dependency graph, which eventually leads to instability. It thus becomes imperative that flows correctly choose their bottlenecks, which is what EMKC achieves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

Corollary 1: Max-min congestion control that converges its



(a) monotonic (b) non-monotonic

Fig. 5. Types of multi-router feedback.



(a) consistent (b) inconsistent

Fig. 6. Spectral radius $\rho(A)$ of system (9)-(10) with $\tau = 0.6$ under 2000 random bottleneck assignments.

feedback $p_l(n)$ at each router 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.

V. 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.

A. Design

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

$$g_l(n) = \frac{\gamma_l C_l - u_l(n)}{N_l(n)}. \tag{9}$$

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

$$x_r(n) = x_r(n - D_r) - \tau\Big(x_r(n - D_r) - g_l(n - D_r^{\leftarrow})\Big), \tag{10}$$

where $\tau > 0$ is the gain parameter. The role of the second term in (10) is twofold. On the one hand, it functions as an efficiency component by encouraging end-users to increase their rates when the resource is under-utilized; on the other hand, it forces the sources to converge their rates to the equal share of the bottleneck link's available capacity so that max-min fairness is achieved in the steady state.

Besides the end-user equation, another important issue is the bottleneck switching mechanism. To this end, each user chooses the link along its path with the largest packet loss $p_l(n)$ as the bottleneck resource, where $p_l(n)$ is based on the *combined* rate $y_l(n) = w_l(n) + u_l(n)$, i.e.,

$$p_l(n) = \frac{y_l(n) - \gamma_l C_l}{y_l(n)}. \tag{11}$$

Since $y_l(n)$ does *not* change when the bottleneck assignment changes (i.e., a flow migrates from one router to another), JetMax is generally monotonic during bottleneck switching.

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

B. 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 $N_l - 1$ other flows, its stationary sending rate is $x_r^* = (\gamma_l C_l - u_l^*)/N_l$, where u_l^* is the steady-state rate of all unresponsive flows at link l.

Proof: In the steady state, we have $x_r(n) = x_r(n - D_r) = x_r^*$ and $u_l(n) = u_l^*$. Combining this with JetMax's end-user equation (10) immediately yields $x_r^* = (\gamma_l C_l - u_l^*)/N_l$. ■

We next show that, under any consistent bottleneck assignment, stability analysis of system (9)-(10) can be reduced to that of EMKC.

Theorem 2: Under any consistent bottleneck assignment, JetMax (9)-(10) is globally asymptotically stable regardless of delay if and only if $0 < \tau < 2$.

Proof: First, assume an undelayed JetMax system with a single link l. Then, its Jacobian matrix A_l is simply $A_l = \text{diag}(1 - \tau)$, 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 [28], we obtain that single-link JetMax is stable for all types of directional and time-varying delay under the same condition on τ. Finally, invoking Theorem 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., $0 < \tau < 2$. Since JetMax (9)-(10) is a linear system, its global stability directly follows. ■

To better understand this theorem, 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 Fig. 6(a) shows, the spectral radius $\rho(A)$ of the system's Jacobian A is $1 - \tau = 0.4$ under all consistent bottleneck assignments, which aligns well with Theorem 2. At the same time, as Fig. 6(b) demonstrates, $\rho(A)$ under inconsistent bottleneck assignments

may exceed 1, in which case even the undelayed system is unstable.

C. 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 routers with zero packet loss in the path of a flow r, it will be uncertain which router 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 router. Assuming that several routers 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

$$b_r = \arg \min_{l \in r: p_l^* = 0} g_l(n). \tag{12}$$

To maintain stability, switching based on the largest packet loss (11) may be performed at any time n; however, that based on (12) is conducted only when flow r's sending rate reaches the ε-neighborhood of stationarity under the current bottleneck assignment. Before proving max-min fairness of the resulting system, we need the following definition.

Definition 1 (Bertsekas-Gallager [2]): A link is a bottleneck of flow i, if it is fully utilized *and* the rate of flow i is no less than that of any other flow accessing the link.

Now we are ready to prove max-min fairness of JetMax.

Theorem 3: The stationary resource allocation of JetMax (10)-(12) is max-min fair.

Proof: Assume 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 router in its path. Let $l \in r$ be the router that provides feedback to flow r. Then, from Lemma 3 we must have $x_r^* = (\gamma_l C_l - u_l^*)/N_l^*$.

Now that link l is fully utilized, 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

$$x_r^* < x_s^*. \tag{13}$$

Let flow s be constrained by router k where $k \neq l$. Then, we have $x_s^* = (\gamma_k C_k - u_k^*)/N_k^*$, which translates (13) into

$$\frac{\gamma_l C_l - u_l^*}{N_l^*} < \frac{\gamma_k C_k - u_k^*}{N_k^*}. \tag{14}$$

According to (12), however, the last inequality must force the bottleneck of flow s to shift from router k to l, thus contradicting the assumption that the system has reached stationarity. ■

D. 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

$$\forall n \geq n_e : \ \frac{y(n)}{\gamma C} \geq 1 - \varepsilon \tag{15}$$

Similarly, $(1-\varepsilon)$-fairness is reached in n_f steps if the system starts in the maximally unfair state and n_f is the smallest integer satisfying

$$\forall n \geq n_f : \frac{|x_r(n) - x_r^*|}{x_r^*} \leq \varepsilon, \ \forall r. \tag{16}$$

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 $\lceil \log_{|1-\tau|} \varepsilon \rceil$ steps.

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 (10) 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

$$y(n) = (1-\tau)^{n/D}(y(0) - \gamma C) + \gamma C, \tag{17}$$

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

$$n_e = \lceil \log_{|1-\tau|} \varepsilon \rceil. \tag{18}$$

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 (10) becomes

$$x_r(n) = x_r(n - D_r) - \tau\Big(x_r(n - D_r) - x_r^*\Big). \tag{19}$$

Solving this recurrence, we get

$$x_r(n) = (1-\tau)^{n/D_r}(x(0) - x_r^*) + x_r^*, \tag{20}$$

which shrinks to $(1-\varepsilon)$-fairness in $n_f = \lceil \log_{|1-\tau|} \varepsilon \rceil$ RTT steps following the technique we used to obtain (18). ■

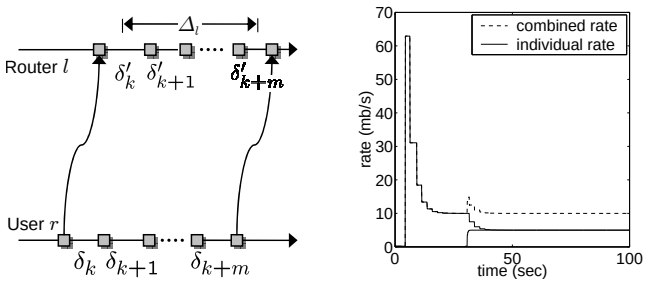
This theorem indicates that JetMax reaches full utilization and converges to fairness over links of *any* capacity in the same number of steps (simulations follow later in the paper). Also observe from (17) and (20) that $0 < \tau < 1$ is required to guarantee monotonicity of the controller. Thus, all JetMax experiments in this paper use $\tau < 1$.

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

VI. Implementation

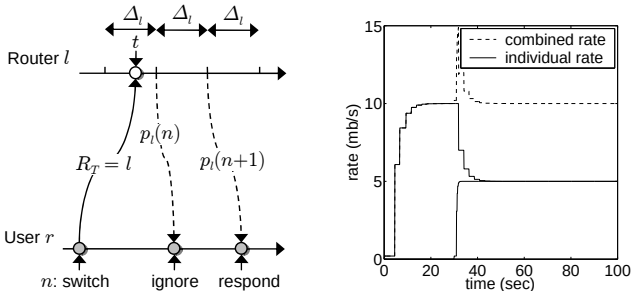
A. Estimating Number of Flows

The first issue encountered by a JetMax router l is how to estimate the current number of responsive flows $N_l(n)$. Our solution to this problem is based on the following observations. For a given flow r, assume that δ_k is the inter-packet departure delay between packets k and $k+1$ at the source and δ'_k is the corresponding inter-packet arrival delay at router l. Fig. 7(a) illustrates this notation and shows that the router's control interval Δ_l generally starts and ends in-between two arriving packets. We therefore have the following relationship between



(a) (b)

Fig. 7. (a) The relationship between control interval Δ_l and inter-packet interval δ_k; (b) JetMax ($\tau = 0.6$ and $\gamma = 1$) with the naive bottleneck switching scheme in $\mathcal{T}_1$.



(a) (b)

Fig. 8. (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 $\mathcal{T}_1$.

the router's control interval and the combined delay of all packets from flow r observed during the interval

$$\sum_{i=k+1}^{k+m-1} \delta'_i \le \Delta_l \le \sum_{i=k}^{k+m} \delta'_i, \tag{21}$$

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

$$\lim_{\Delta_l \to \infty} \frac{\sum_{i=k}^{k+m} \delta'_i}{\Delta_l} = \lim_{\Delta_l \to \infty} \frac{\sum_{i=k+1}^{k+m-1} \delta'_i}{\Delta_l} = 1. \tag{22}$$

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

$$\lim_{\Delta_l \to \infty} \frac{\sum_{r=1}^{N_l} \sum_{i=k}^{k+m} \delta'_i}{\Delta_l} = N_l. \tag{23}$$

Even though in general δ_k does not equal to δ'_k, sums of these two metrics over a large number of packets are asymptotically equal, i.e., $\lim_{m\to\infty} \sum_{i=k}^{k+m} \delta_i = \lim_{m\to\infty} \sum_{i=k}^{k+m} \delta'_i$, since JetMax does not build up network queues or lose any packets. This, combined with (23), leads to

$$\lim_{\Delta_l \to \infty} \frac{\sum_{r=1}^{N_l} \sum_{i=k}^{k+m} \delta_i}{\Delta_l} = N_l. \tag{24}$$

Using the last equation, we next develop a mechanism for estimating N_l. Each user r includes in every packet k its inter-packet departure delay $\delta_k = s_k / x_r(n)$, where s_k is the size of the packet and $x_r(n)$ is the current sending rate. The router then sums up this field over all packets of all *responsive* flows and averages this value over interval Δ_l. From (24), we have that the value $\tilde{N}_l = \sum_{r=1}^{N_l} \sum_{i=0}^{m} \delta_{k+i} / \Delta_l$ converges to the true number of flows N_l as Δ_l grows to infinity. Note that this method does *not* maintain state information about individual flows and requires only one addition per arriving packet and one division per interval Δ_l.

B. 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.

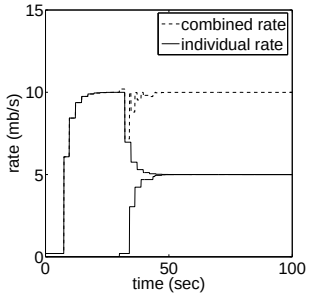
Upon each packet arrival, router l increments R_C by one and then checks whether its local packet loss $p_l(n)$ is greater than the one carried in the packet. If both packet-loss values are zero, the router checks if its local average rate $g_l(n)$ is less than the one carried in the header. If either 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.

C. Managing Bottleneck Switching

The above scheme in itself is insufficient to eliminate all undesirable transient effects associated with bottleneck switching. To demonstrate this, we simulate the algorithms developed so far in `ns2` using the single-link topology $\mathcal{T}_1$, where we change the join order of users to highlight some of the issues arising in the naive implementation of JetMax. Specifically, flows x_2 and x_1 join at time 0 and 30 seconds, and experience round-trip delay 2020 and 220 ms, respectively. The simulation result is plotted in Fig. 7(b), in which x_2 initially overflows the link's capacity by 500% and then maintains non-zero packet loss for over 15 seconds. As we discuss below, this phenomenon arises as the result of improper management of bottleneck switching.

For the illustration in Fig. 8(a) that explains this situation, assume that user r changes its bottleneck to link l at time n and the first packet carrying this new membership arrives into router l at time $t = n + D^{\rightarrow}_{r,l}$, which is in the middle of the router's control interval Δ_l. Notice that flow r is counted as *unresponsive* prior to time t and *responsive* after that. This inconsistent inference of membership results in 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



(a)

Source port #		Dest port #	
Flags	R_T	R_C	R_S
User sequence #			
Packet loss: p_l^+			
Fair rate: g_l^+			
Proposed size: s_k^+			
Inter-packet interval: δ_k			

32 bits

(b)

Fig. 9. (a) JetMax ($\tau = 0.6$ and $\gamma = 1$) with proposed rate in $\mathcal{T}_1$; (b) format of the JetMax packet header.

Δ_l after the switch. Thus, to properly manage bottleneck switching, the end-user simply ignores the first non-duplicate ACK after each switch and reacts to the following ones as shown in Fig. 8(a). Simulation of the resulting JetMax is illustrated in Fig. 8(b), in which the initial "spike" present in Fig. 7(b) is eliminated and x_2 monotonically converges to efficiency. However, notice in the figure that JetMax exhibits transient packet loss reaching as high as 33% when flow x_1 joins the network. We explain and resolve this issue in the next subsection.

D. Eliminating Transient Packet Loss

The reason of the transient packet loss shown in Fig. 8(b) lies in the fact that flow x_2 with a large RTT does not release bandwidth quickly enough and is not aware of the presence of any competing flows until after the overshoot has happened.

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

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

$$s_k^+ = s_k \frac{x_r^+(n)}{x_r(n)}. \tag{25}$$

By adding up the virtual packet sizes and normalizing them by the interval length Δ_l, the router can approximate the aggregate proposed rate $y_l^+(n) = \sum_{r \in l} x_r^+(n)$ and thus accept or decline $y_l^+(n)$ at the end of its control interval Δ_l based on whether $y_l^+(n)$ is greater than $\gamma_l C_l$ or not. Note that when computing $g_l(n)$ in (9) and $p_l(n)$ in (11), the router simply replaces $u_l(n)$ and $y_l(n)$ with their corresponding proposed values $u_l^+(n)$ and $y_l^+(n)$. Therefore, no extra latency is introduced by this mechanism and each approved rate adjustment takes exactly one RTT (instead of two RTTs if (9)-(11) were based on actual rates). The result of this implementation is shown in Fig. 9(a), in which the system never overflows the link and converges to fairness monotonically.

E. Calculating Reference Rate

Similar to the discussion of bottleneck switching in Section VI-C, an inconsistency between the router's and the end-user's reference rates arises when packets carrying the new proposed rate $x_r^+(n)$ arrive in the middle of the router's control interval Δ_l (see [28] for details). As proposed in [28], this inconsistency can be resolved by utilizing the packet sequence number and keeping track of the transmitted packets at the source to recover the reference rate used by the router. However, this results in significant computational overhead and in certain cases may adversely impact the ability of the sender to maintain high sending rates.

Another problem of the above method from [28] lies in the fact that the obtained reference rate is a function of the previous and current proposed rates. As the 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 bottleneck, oscillations.

A much simpler approach that also significantly improves the performance of reference rate calculation from [28] is to leverage the fact that this inconsistency exists only in the *first* control interval after the switch. Thus, by ignoring the first non-duplicate ACK after the switch and responding to the remaining ones, 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 $x_r^+(n - D_r)$ in its JetMax equation (10) to compute the next proposed rate $x_r^+(n)$.

F. Packet Format

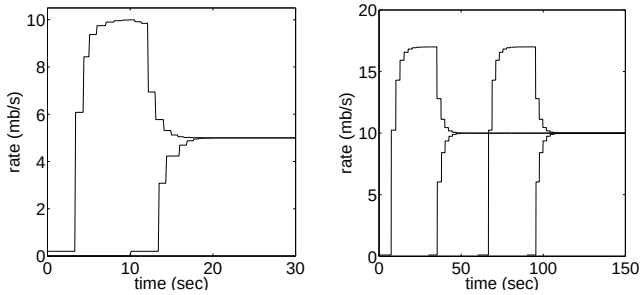
The header format of a JetMax packet is illustrated in Fig. 9(b). Besides the two-byte fields for port numbers, we allocate a one-byte field to each of *flags*, R_T, R_C, and R_S. Then, we use four-byte numbers to record the user sequence numbers to deal with out-of-order packets, packet loss $p_l^+ = 1 - \gamma_l C_l / y_l^+$ computed based on the proposed rates, fair rate $g_l^+ = (\gamma_l C_l - u_l^+)/N_l$ also based on the proposed rates, user-proposed packet size s_k^+, and the inter-packet interval $\delta_k = s_k / x_r(n)$. Note that only δ_k uses the actual sending rate of the flow.

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

VII. Simulations

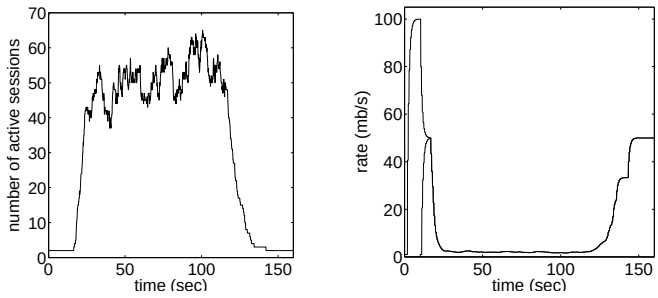
A. Behavior in $\mathcal{T}_1$ and $\mathcal{T}_2$

We first repeat the `ns2` simulations that earlier presented stability and equilibrium problems to existing methods and



(a) $\mathcal{T}_1$ with random delay (b) $\mathcal{T}_2$

Fig. 10. Performance of JetMax ($\tau = 0.6$ and $\gamma = 1$) in ns2.



(a) number of active sessions (b) dynamics of long flows

Fig. 11. Single-link performance of JetMax ($\tau = 0.6$ and $\gamma = 1$) in the presence of mice flows.

then examine how JetMax handles additional scenarios. Simulation code used in this paper is available in [10].

JetMax's single-link performance under *constant* heterogeneous delay has already been shown in Fig. 9(a). We next create a more complex dynamic system in which the forward and backward feedback delays are non-deterministic and time-varying. We first modify $\mathcal{T}_1$ by setting the round-trip propagation delay of each link to 20 ms. We then 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 packets by 10 μs, where d and m are uniformly distributed in $[0.5, 1.0]$ and $[5000, 10000]$, respectively. This delay pattern ensures that the queue is completely emptied before the next spike and approximates periodic congestion in the Internet caused by flash crowds, routing changes, and oscillatory behavior of cross-traffic flows. Simulation results under this configuration are plotted in Fig. 10(a), in which JetMax is stable, max-min fair, and loss-free as expected. It is also worthwhile to note that the flat regions in Fig. 10(a) when both flows start consume three RTTs (i.e., 2.6 seconds) and are necessary for the flows to deal with initial router assignment and bottleneck selection.

Multi-link performance of JetMax in $\mathcal{T}_2$ is shown in Fig. 10(b), in which the protocol again demonstrates monotonic convergence, max-min allocation of rates in the steady state, and effective handling of bottleneck selection. Numerical data from this simulation 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.

B. Effect of Mice Traffic

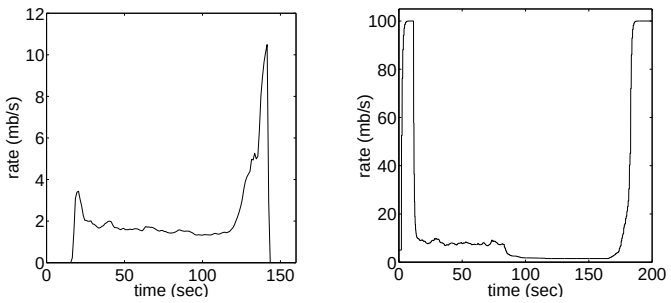
All of our simulations so far have been performed in environments with long-lived flows. However, the real Internet traffic is composed of a mixture of connections with a wide range of transfer sizes, packet sizes, and RTTs [8]. Thus, to obtain a better understanding of JetMax, we next test it in more diverse scenarios.

Toward this end, we first consider a simple "dumb bell" topology, where 2 long and 500 short JetMax flows share a single link of capacity 100 mb/s.The inter-arrival time of short flows follows an exponential distribution with mean $\lambda = 0.2$ seconds and the duration of each flow is drawn from a log-normal distribution [20] with mean $\omega = 10$ seconds. From basic queuing theory, we can infer that the expected number of active short flows at any instant is $L = \omega/\lambda = 50$, while the instantaneous flow population is bursty as illustrated in Fig. 11(a). Moreover, we set the packet sizes of the short flows to be uniformly distributed in $[800, 1300]$ bytes and their RTTs are selected uniformly randomly in $[40, 1040]$ ms.

As seen in Fig. 11(b), one long flow starts first and quickly reaches link utilization. After the second long flow joins 5 seconds later, the first flow is forced to release some of its bandwidth, allowing both flows to converge to the fair share of the link's capacity (i.e., 50 mb/s). At time 15 seconds, mice flows start joining and leaving the network. Since on average there are 50 short and 2 long flows in the system, the *expected* fair rate is $100/52 = 1.92$ mb/s per flow. This prediction is confirmed in Fig. 11(b), where the sending rates of the long flows remain within $[1.7, 2.0]$ mb/s during the period between $[30, 120]$ seconds. It is worth noting that the small rate oscillations during this interval are not due to instability, but the time-varying number of mice flows and changes to the stationary point of the system.

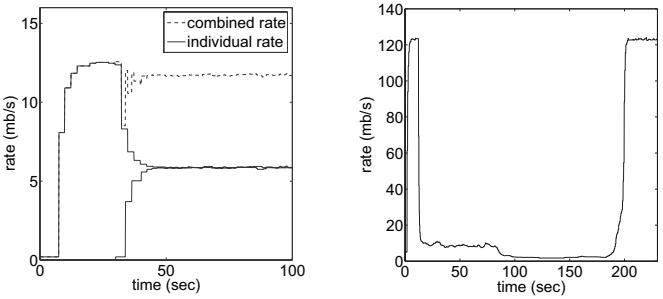
To understand the throughput obtained by the short flows, Fig. 12(a) shows the average rate of the mice flows. As seen in the figure, the short flows also manage to obtain their fair share (despite the short duration) and achieve rates close to the expected 1.92 mb/s. As the number of active connections decreases after time 120 seconds, sending rates of the remaining short flows climb up and take over the bandwidth of the departed flows.

We 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 (router control intervals Δ_l are uniformly random in $[100, 300]$ ms), each of which is accessed by 500 short flows. As shown in Fig. 12(b), the long flow starts first and converges to the capacity of R_2. Short flows accessing R_1 start joining the system after 15 seconds. Since R_1 becomes more congested than R_2, the long flow switches the bottleneck to R_1 and maintains its sending rate within the neighborhood of the average fair rate $400/52 = 7.7$ mb/s. At time 80 seconds,



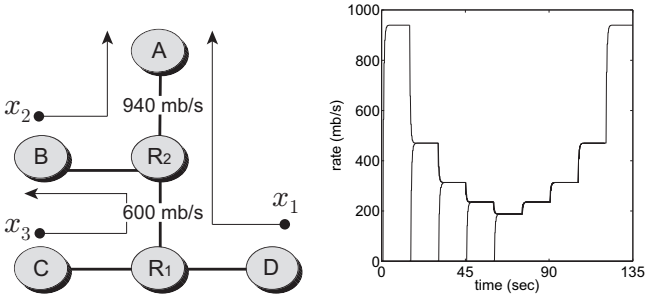
(a) (b)

Fig. 12. (a) Dynamics of the average throughput of short flows in a single-link network; (b) Multi-link performance of the long JetMax flow ($\tau = 0.6$ and $\gamma = 1$) in the presence of mice traffic.



(a) (b)

Fig. 13. JetMax ($\tau = 0.6$ and $\gamma = 1$) under random packet loss: (a) $\mathcal{T}_1$ with 10% forward-path loss; (b) "parking lot" topology with mice flows and random loss.



(a) topology (b) single-router test

Fig. 14. Single-router Linux experiments with JetMax ($\tau = 0.6$).

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.

C. Effect of Random Packet Drops

In this subsection, we examine the performance of JetMax in lossy environments (e.g., wireless networks) with random non-congestion-related packet drops. We first note that JetMax is not sensitive to packet loss in the return path since out of the ACKs generated in the same Δ_l interval, only one is utilized by the end-user to adjust its sending rate and all others are ignored since they carry duplicate information. We verified this in `ns2` simulations, where the performance of JetMax in $\mathcal{T}_1$ with 90% packet loss in its return path was almost identical to that in the loss-free environment previously shown in Fig. 9(a). We omit the plot of this simulation for brevity and focus on more interesting cases of forward-path loss.

To better see the effect of random loss in the forward path, consider the `ns2` simulation illustrated in Fig. 13(a), where we use $\mathcal{T}_1$ and create 10% and 20% packet loss in the forward paths of flows x_1 and x_2, respectively. As shown in the figure, both fairness and stability are not affected by the forward-path random loss; however, the stationary rates are. To explain this phenomenon, assume $1 - \alpha_{r,l}$ is the total (long-term average) packet loss suffered by flow r along its path to router l. Using Lemma 3, it is not difficult to obtain that

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

where the average loss rate α_l is given by

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

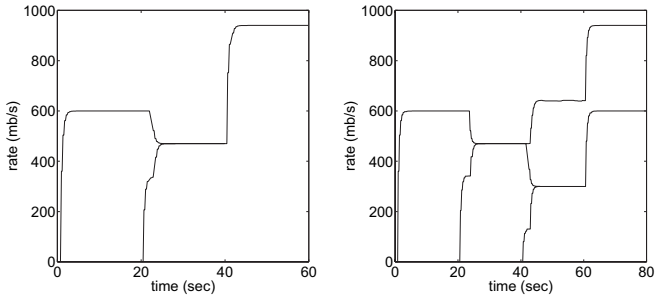
and S_l is the set of responsive flows with respect to link l. Accordingly, we have that the stationary rate x_1^* before the second flow joins the network is $10/0.8 = 12.5$ mb/s, while afterwards both x_1^* and x_2^* are $5/0.85 = 5.82$ mb/s, all of which matches simulation results perfectly. Since 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)-(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. 12(b) with 500 mice flows per link, 10% random loss on each link in the forward path, and 50% loss in the backward path. Fig. 13(b) shows the dynamics of the long flow and confirms that JetMax is stable and convergent to the equilibrium as expected.

VIII. 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 [27] 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. 14(a), where R_1 is a single Pentium 4 running at 3.4 GHz and R_2 is a dual-Xeon box running at 3 GHz. All network cards are 1 gb/s full-duplex 1000BaseT Ethernet utilizing PCI-X slots in the their respective computers. Network capacity in the figure is in terms of *transport-layer* rates and is configured independently for each link at 600 and 940 mb/s using different target utilization levels γ_l.

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



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

Fig. 15. Dual-router Linux experiments with JetMax ($\tau = 0.6$).

point operations (despite a popular belief to the contrary [27]) and that kernel timers are scheduled with remarkable accuracy (i.e., 100 μs), both of which provide significant benefit to AQM algorithms as they often require computation of feedback with high precision and accurate Δ-interval timing.

For the first test, we run five flows from host B to A in Fig. 14(a) to examine the ability of JetMax to utilize high-bandwidth links and support multiple senders/receivers per end-host. Each flow starts with a 15-second delay and lasts for 75 seconds. The performance of JetMax for this setup is shown in Fig. 14(b). Notice in the figure that the first flow converges to 99% of 940 mb/s in 1.3 seconds and maintains its steady-state rate without oscillations. As subsequent flows arrive, they take 1.2 seconds (which is 6 control steps of $\Delta = 200$ ms units each) to achieve 0.99-fairness, where transitions between the neighboring states take place monotonically and the system's combined rate never exceeds 940 mb/s. Similar performance is observed when flows depart, where the system takes approximately 1.2 seconds to 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. 14(a) with a 20-second delay. Notice that x_1 should first converge to 600 mb/s, then shift its bottleneck to R_2, and eventually settle down at 470 mb/s. This is shown in Fig. 15(a), where the flows perform precisely as expected. When flow x_1 departs at $t = 40$, x_2 quickly converges to 940 mb/s.

In our final setup, we repeat the same experiment except that flow x_3 joins at time $t = 40$ seconds. This allows the bottleneck of flow x_1 to shift twice during its stay in the system. The corresponding simulation result is illustrated in Fig. 15(b), where x_1 and x_2 first converge to 470 mb/s each and maintain this rate until $t = 40$. When x_3 joins, it quickly settles down with x_1 at 300 mb/s and x_2 takes the remaining bandwidth (i.e., 640 mb/s) on its link. 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.

IX. 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.

References

[1] H. Balakrishnan, N. Dukkipati, N. McKeown, and C. Tomlin, "Stability Analysis of Switched Hybrid Time-Delay Systems – Analysis of the Rate Control Protocol," Tech. Report, Stanford University, Jul. 2004.

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

[3] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith, "Tuning RED for Web Traffic," *ACM SIGCOMM*, Aug. 2000.

[4] R. DeCarlo, M. S. Branicky, S. Pettersson, and B. Lennartson, "Perspectives and Results on the Stability and Stabilizability of Hybrid Systems," *Proceedings of the IEEE*, 88(2), Jul. 2000.

[5] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor Sharing Flows in the Internet," *IWQoS*, June 2005.

[6] A. Falk and D. Katabi, "Specification for the Explicit Control Protocol (XCP)," Tech. Report, USC/ISI, Oct. 2005.

[7] S. Floyd, "High-speed TCP for Large Congestion Windows," *IETF RFC 3649*, Dec. 2003.

[8] S. Floyd and E. Kohler, "Internet Research Needs Better Models," *HotNets-I*, Oct. 2002.

[9] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong, "On Designing Improved Controllers for AQM Routers Supporting TCP Flows," *IEEE INFOCOM*, Apr. 2001.

[10] JetMax@TAMU, `http://irl.cs.tamu.edu/projects/mkc/`.

[11] C. Jin, D. Wei, and S. H. Low, "FAST TCP: Motivation, Architecture, Algorithms, Performance," *IEEE INFOCOM*, Mar. 2004.

[12] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth Delay Product Networks," *ACM SIGCOMM*, Aug. 2002.

[13] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability," *Journal of the Operational Research Society*, 49(3):237–252, Mar. 1998.

[14] S. Kunniyur, "AntiECN Marking: A Marking Scheme for High Bandwidth Delay Connections," *IEEE ICC*, May 2003.

[15] S. Kunniyur and R. Srikant, "Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management," *ACM SIGCOMM*, Aug. 2001.

[16] S. Kunniyur and R. Srikant, "Stable, Scalable, Fair Congestion Control and AQM Schemes that Achieve High Utilization in the Internet," *IEEE Trans. on Automatic Control*, 48(11):2024–2029, Nov. 2003.

[17] S. Liu, T. Basar, and R. Srikant, "Pitfalls in the Fluid Modeling of RTT Variations in Window-Based Congestion Control," *IEEE INFOCOM*, Mar. 2005.

[18] S. H. Low, L. L. H. Andrew, and B. P. Wydrowski, "Understanding XCP: Equilibrium and Fairness," *IEEE INFOCOM*, Mar. 2005.

[19] L. Massoulié, "Stability of Distributed Congestion Control with Heterogeneous Feedback Delays," *IEEE/ACM Trans. on Networking*, 47(6):895–902, Jun. 2002.

[20] V. Paxson, "Empirically Derived Analytic Models of Wide-Area TCP Connections," *IEEE/ACM Trans. on Networking*, 2(4):316–328, Aug. 1994.

[21] R. S. Prasad, M. Jain, and C. Dovrolis, "On the Effectiveness of Delay-Based Congestion Avoidance," *PFLDnet*, Feb. 2004.

[22] K. K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," *IETF RFC 3168*, Sep. 2001.

[23] J. Wang, D. X. Wei, and S. H. Low, "Modeling and Stability of FAST TCP," *IEEE INFOCOM*, Mar. 2005.

[24] B. P. Wydrowski and M. Zukerman, "MaxNet: A Congestion Control Architecture for Maxmin Fairness," *IEEE Communication Letters*, 6(11):588–599, Nov. 2002.

[25] XCP@ISI, `http://www.isi.edu/isi-xcp/`.

[26] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One More Bit is Enough," *ACM SIGCOMM*, Aug. 2005.

[27] Y. Zhang and T. Henderson, "An Implementation and Experimental Study of the eXplicit Control Protocol (XCP)," *IEEE INFOCOM*, Mar. 2005.

[28] Y. Zhang, S.-R. Kang, and D. Loguinov, "Delayed Stability and Performance of Distributed Congestion Control," *ACM SIGCOMM*, Aug. 2004.