Computer Networks 53 (2009) 1027–1039

Contents lists available at ScienceDirect

Computer Networks

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

Towards experimental evaluation of explicit congestion control

Yueping Zhang [a,*], Saurabh Jain [b], Dmitri Loguinov [c]

[a] NEC Laboratories America, Inc., Princeton, NJ 08540, United States
[b] Cisco Systems, Inc., San Jose, CA 95134, United States
[c] Texas A&M University, College Station, TX 77843, United States

ARTICLE INFO

Article history:
Received 26 March 2008
Received in revised form 8 August 2008
Accepted 8 December 2008
Available online 25 December 2008

Responsible Editor: X.S. Shen

Keywords:
Congestion control
Explicit-feedback
Linux experiment

ABSTRACT

Innovative efforts to provide a clean-slate design of congestion control for future high-speed heterogeneous networks have recently led to the development of explicit congestion control. These methods rely on multi-byte router feedback and aim to contribute to the design of a more scalable Internet of tomorrow. However, experimental evaluation and deployment experience with these approaches is still limited to either low bandwidth networks or simple topologies. In addition, all existing implementations are exclusively applicable to either rate- or window-based protocols and are unable to study performance of different protocols on a common platform. This paper aims to fill this void by designing a unified Linux implementation framework for both rate- and window-based methods that does not incur any interference with the system's network stack or applications. Using this implementation, we implement and reveal several key properties of four recent explicit congestion control protocols XCP, JetMax, RCP, and PIQI-RCP using Emulab's gigabit testbed with a ariety of simple yet representative network topologies. Our experiments not only confirm the known behavior of these methods, but also demonstrate their previously undocumented properties (e.g., RCP's transient overshoot under abrupt traffic load changes and JetMax's low utilization in the presence mice flows).

1. Introduction

The explosive evolution of the Internet in both its size and capacity has given rise to serious concerns [12,21,25,36] in the research community about the effectiveness of existing congestion control methods. As a consequence, significant collaborative research initiatives (e.g., GENI [14] and FIND [11]) are currently under way to design and develop a materially different Internet that fundamentally improves scalability, efficiency, and robustness of the current Internet. Driven by these initiatives, recent efforts to design next-generation congestion control have led to the emergence of explicit-feedback control [6,19,34,35,39,41], which significantly improves traditional end-to-end methods [1–5,12,13,15,21,22,24,25,30,36] in terms of stability, scalability, and fairness.

Explicit congestion control involves multi-byte congestion notification from network devices and uses this information in the control loop of end-users. Some of the proposed protocols in this category are XCP [19], MaxNet [34,35], EMKC [39], RCP [6], and JetMax [41]. Theoretical and extensive `ns2` [31] simulation results of XCP, RCP, and JetMax indicate that they can provide a congestion-control framework that achieves scalability in terms of bandwidth and round-tip propagation delay, exhibits stability, and delivers inter-flow fairness. However, experimental studies with these methods are limited to just three papers: XCP over a single 10-mb/s bottleneck [37], MaxNet in a WAN topology with a single 10-mb/s bottleneck [33], and JetMax in three scenarios with multiple gigabit bottlenecks [40,41].

* Corresponding author. Tel.: +1 609 951 2647; fax: +1 609 951 2480.
E-mail addresses: yueping@nec-labs.com (Y. Zhang), saujain@tamu.edu (S. Jain), dmitri@cs.tamu.edu (D. Loguinov).

In this paper, we aim to advance research in explicit congestion control by implementing a unified Linux framework of explicit congestion control for both window-based and rate-based methods and conducting a simple, yet informative, comparison study using the developed software in both single and multi-link topologies in real systems and gigabit networks. Our implementation can be applied to design and testing of any new explicit congestion control schemes, standardization of existing methods, porting of these algorithms to other operating systems, and experimentation in higher-capacity networks and a wider variety of network topologies and traffic conditions.

To achieve our goal, we first implement a general explicit-feedback congestion-control framework in the Linux TCP/IP stack and populate it with four protocols XCP, JetMax, and RCP. We also include in our study a recently proposed method called *Proportional-Integral Queue-Independent RCP* (PIQI-RCP),[1] which addresses the drawbacks of RCP (i.e., instability in certain networks and large buffer requirements during flow join [17]). Conventionally, rate-based protocols are implemented as application-level libraries using UDP [41] or a combination of UDP and TCP connections [32]. In contrast, we develop in this paper a methodology for integrating rate-based protocols (e.g., RCP, JetMax, PIQI-RCP) into the window-based TCP/IP stack, which allows existing TCP-based applications to use them without modification. In addition, this methodology leads to a unified platform that facilitates fair evaluation of rate-based methods in comparison to their window-based counterparts (e.g., XCP).

Using our implementation, we then perform a series of Linux experiments to study the behavior of these protocols and verify that our implementation matches `ns2` simulations shown in the original papers. Our results demonstrate that these methods can indeed achieve high link utilization, RTT-independent fairness (which is missing in most end-to-end methods [36]), and zero steady-state queueing delay and packet loss in both single and multi-link topologies. Our experiments also lead to several novel findings: (a) RCP can be much slower in its convergence rate to the equilibrium compared to the other three methods; (b) JetMax loses link utilization in the presence of very short (i.e., sub-RTT in duration) flows; and (c) RCP exhibits oscillatory behavior with large overshoots and undershoots during abrupt changes in traffic load. We finish the paper by explaining the cause of these drawbacks and discussing the tradeoffs of the studied methods.

1.1. Contributions of the paper

We note that all protocols examined in the paper has been extensively studied by their original authors using `ns2` simulations. However, in addition to proposing a novel unified implementation framework and reporting several newly discovered behaviors of these protocols, our contributions also lie in the fact that we are able to address many fundamental questions that `ns2` simulations cannot answer.

(a) *Precise timestamps:* It has been a long standing issue that the ideal timing of `ns2` would not be possible to replicate in practice and therefore Active Queue Management (AQM) congestion control would not be able to compute its metrics (e.g., incoming rates) accurately in real routers. Our work has shown that this is not the case *even* with tiny 10-ms sampling windows and relatively slow Linux software routers running a generic OS.

(b) *Precise packet scheduling:* `ns2` can transmit packets with arbitrary inter-packet delays; however, real networks cannot do that. Not only does this occur due to random OS scheduling delays, but also interrupt moderation [18]. Our work shows that even if sending rates are fluctuating on very small time-scales, all studied AQM techniques are robust enough to precisely estimate not only the average arrival rates, but also to compute such complex metrics as the number of responsive flows, virtual packet loss, and smoothed end-to-end RTT.

(c) *Performance:* It has been argued that AQM congestion control is too burdensome for real routers and is not viable in high bandwidth networks. `ns2` simulations cannot be used to verify or disprove this conjecture. Our results show that per-packet processing of most studied methods is very small and can be accommodated even in software routers of gigabit networks with virtually no additional CPU overhead.

(d) *Contribution to the community:* By sharing a working implementation and explaining the details of how it can be performed, one can actively promote the deployment of new protocols in real networks. On the other hand, `ns2` simulation code is often too detached from reality and cannot be easily adopted into existing kernels and architectures.

The rest of the paper is organized as follows: in Section 2, we briefly review the theory behind XCP, JetMax, RCP, and PIQI-RCP. In Section 3, we discuss our implementation framework and relevant modifications to the Linux kernel for implementing these methods. In Section 4, we verify our developed code and evaluate the performance of these protocols in various scenarios. We conclude the paper in Section 5.

2. Background

In explicit congestion control, network devices in the path feed back multi-byte congestion information to end-hosts so that the latter can timely and accurately adjust their congestion window sizes or sending rates in response to the current traffic situation in the network. Each router l performs per-packet processing to compute the combined input traffic rate and generate feedback signal $p_l(t)$. In this paper, we focus on protocols relying on *max–min* feedback, where each flow i with round-trip time (RTT) D_i responds only to feedback $p_l(t)$ from the most congested router l in its path, which is called the *bottleneck* router of user i. The forward/backward delays of flow i to/from the bottleneck router l is denoted by $D_i^{\rightarrow}$ and $D_i^{\leftarrow}$, respectively.

Since its appearance in 2002, *eXplicit Control Protocol* (XCP) [19] has become a de-facto standard for explicit con-

[1] Pronounced "Picky RCP".

gestion 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. Each router employs a decoupled efficiency controller (EC) and fairness controller (FC). Specifically, EC generates the desired *aggregate* change in the congestion window of all flows:

$$\phi(t) = \alpha d S(t) - \beta Q(t), \tag{1}$$

where α and β are constants, d is the average RTT, $S(t)$ is the available bandwidth, and $Q(t)$ is the persistent queue size at the bottleneck link. FC then translates $\phi(t)$ into per-packet feedback $H_i(k)$, which is conveyed in the k-th ACK of flow i. Upon arrival of each ACK, flow i sets its congestion window $W_i(k)$ according to:

$$W_i(k) = \max(W_i(k-1) + H_i(k), s), \tag{2}$$

where s is the packet size. For homogeneous delay, it is shown in [19] that XCP is stable in single-bottleneck topologies if $0 < \alpha < \pi/4\sqrt{2}$ and $\beta = \alpha^2\sqrt{2}$, where α and β are constants used in the XCP control equation.

The remaining three schemes examined in the paper are all rate-based. Specifically, in JetMax [41], flow i adjusts its sending rate $x_i(t)$ using the following control equation:

$$x_i(t) = x_i(t - D_i) - \tau(x_i(t - D_i) - g_l(t - D_i^{\leftarrow})), \tag{3}$$

where $0 < \tau < 2$ is the gain parameter and $g_l(t)$ is the network feedback, which can be interpreted as the estimated fair rate at the bottleneck link:

$$g_l(t) = \frac{\gamma_l C_l - u_l(t)}{N_l(t)}, \tag{4}$$

where γ_l is the desired link utilization, $N_l(t)$ is the total number of flows bottlenecked at l, and $u_l(t)$ is the aggregate rate of flows receiving feedback from routers other than l. It is proven in [41] that JetMax achieves delay-independent stability, max–min fairness, zero packet loss, and constant convergence speed to both efficiency and fairness.

The recently proposed Rate Control Protocol (RCP) [6] is a rate-based max–min Active-Queue Management (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

$$r_l(t) = r_l(t - \Delta)\left[1 - \frac{\Delta}{d_l C_l}\left(\alpha(y_l(t) - C_l) - \beta\frac{q_l(t)}{d_l}\right)\right], \tag{5}$$

where Δ is the router's control interval, α and β are constants, d_l is a moving average of RTTs sampled by link l, C_l is its capacity, and $q_l(t)$ is its queue length at time t. Compared to XCP, RCP has lower implementation overhead, offers quicker transient dynamics, and achieves max–min fairness [6].

However, it is recently demonstrated that RCP suffers instability in certain network topologies and prohibitively large buffer requirement when flows dynamically join the network [17]. To address these issues, a new protocol called *Proportional Integral Queue Independent RCP* (PIQI-RCP) is proposed in [17]. In PIQI-RCP, each router l employs a Proportional Integral (PI) controller [27] to compute the feedback $R_l(t)$:

$$R_l(t) = R_l(t - T)[1 + \kappa_1 e_l(t) + \kappa_2 e_l(t - T)], \tag{6}$$

where T is the control interval, $e_l(t) = 1 - y_l(t)/(\gamma_l C_l)$ is the error function of router l, C_l is its link capacity, and κ_1, κ_2 are constants. At the sending side, each flow i applies the following control equation to adjust its sending rate $x_i(t)$:

$$x_i(t) = x_i(t - T) + \tau_1 e_i(t) + \tau_2 \delta_i(t), \tag{7}$$

where:

$$e_i(t) = R_l(t - D_i^{\leftarrow}) - x_i(t - T) \tag{8}$$

is the error between the previous rate of user i and the rate suggested by the network,

$$\delta_i(t) = R(t - D_i^{\leftarrow}) - R_l(t - T - D_i^{\leftarrow}) \tag{9}$$

is the difference between the two most-recent rates provided by router l to flow i, and τ_1, τ_2 are constants. Assuming $\alpha_1 = \alpha_2 = \alpha$, PIQI-RCP has been proven to be stable in a single-bottleneck topology for flows with both homogeneous and heterogeneous RTTs [17].

An additional explicit-feedback methods are inspired by Kelly's optimization framework [20] and aim to improve stability and convergence properties of traditional models of additive packet loss [23]. This method is called Exponential Max–min Kelly Control (EMKC) [39], 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. Although EMKC is proven to achieve delay-independent stability and exponential convergence rate to efficiency, it is demonstrated in [38] to suffer consistent packet loss in the steady state. Thus, we do not include EMKC in this study.

3. Implementation details

The first step in any experimental evaluation of congestion control protocols is to design an efficient implementation that can perform well in high-speed networks. We present our implementation details in this section and provide experimental results in the following section.

3.1. General caveats

For our implementation, we used Linux kernel 2.6.12, which is available with the Fedore Core 4 Linux distribution [9]. To generate accurate feedback information, unlike old versions [37], Linux now supports floating-point computations, which can be enabled in the `Makefile` during re-compilation of the kernel. An additional issue with kernel-space floating-point calculations is that starting from Linux kernel 2.6.0, kernel threads can be preempted, in which case floating-point registers are not saved due to additional memory overheads. To circumvent this preemption problem, portions of floating-point code must be protected, which is accomplished by locking the CPU using the `get_cpu()` and `put_cpu()` kernel routines.

Our implementation of the end-host and AQM-router functionality is in the form of kernel driver modules, which can be loaded/unloaded without rebooting the system. Thus, the following proposed framework is not limited to protocols examined in the paper (i.e., RCP, PIQI-RCP, Jet-Max, and XCP), but can be easily applied to any other explicit congestion control by modifying the end-host/router kernel modules according to the protocols' control equations. Since no change to Linux' TCP/IP stack is required, the resulting protocols can be conveniently installed and tested across different machines with different Linux platforms. We next start our discussion with implementation detail of the end-host.

3.2. End-host

In all explicit-feedback protocols, the end-host uses feedback explicitly calculated by intermediate routers to adjust its sending rate (or congestion window). To facilitate routers' feedback computation, the end-host may also need to convey in the packet header certain congestion-related information (e.g., the current RTT estimate, congestion window, sending rate, and inter-packet interval). Inspired by the discussion in [8], we place between each packet's transport and network headers a new congestion (AQM) layer, which is used to communicate congestion-related information and network feedback between end-hosts and routers. Congestion headers can be conveniently inserted into (or removed from) every outgoing (or incoming) packet by the end-host module without interfering with the operation in the TCP/IP layer. Thus, this methodology is versatile enough to be easily implemented in different operating systems without modifying their TCP/IP stacks. In addition to the congestion header, TCP options [33,37] and IP options [8] have also been suggested to communicate congestion-related information and feedback. We refer interested readers to [8] for a comprehensive discussion of available strategies.

We next provide a detailed description of the end-host module. We first assign each protocol a different protocol number, which is used by the end-host module to register (or unregister) with the underlying TCP/IP stack every time it is loaded (or unloaded) in the kernel. As illustrated in Fig. 1, every incoming packet from the Network Interface Card (NIC) driver is first passed to the IP layer for processing. Depending upon the protocol number associated with the packet, the IP layer forwards the packet to the corresponding AQM module that has registered for it. Then, the sender module extracts feedback information from the congestion header and invokes its control equation to calculate the next sending rate or congestion window. If the end-host is a data receiver, the module saves the feedback in the congestion header and echoes it back to the sender as part of acknowledgement packets.

Analogously, every outgoing packet from the application layer is first processed by the TCP layer, which delegates packet transmission to the corresponding end-host module for appending congestion header and filling in congestion-related information. The packet is then handed over by the end-host module to the IP layer for further processing and finally passed to the NIC driver for transmission. Since most of the operations mentioned above are performed inside the kernel-space, they are transparent to applications. Thus, this design facilitates deployment of these protocols in the future Internet without requiring any changes to existing applications.

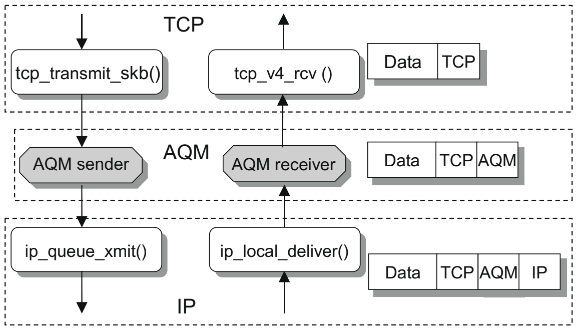

Fig. 1. Implementation methodology of explicit-feedback congestion control protocols.

Since most operations mentioned above are done inside the core kernel and modules, they are transparent to the applications. This design facilitates deployment of protocols in future networks without requiring any change to current applications. We next provide more details about the implementation of window-based and rate-based schemes for end-hosts.

For window-based methods (e.g., XCP), minimal changes in the current TCP/IP stack are required since TCP itself is window-based. For rate-based schemes, however, additional modifications in the kernel are necessary to make its original window-based data-transfer operation rate-based. The idea is very simple – pace the transmission of data packets at the rate computed by the end-host module's control equation. To accomplish this goal, we first seek to understand how data is transmitted in the original TCP/IP stack. Data sent by the application layer is sliced into chunks of MSS (Maximum Segment Size) and queued into a buffer by the transport layer, which appends its headers to packets upon their departure. The normal behavior is to transmit them *instantaneously* when the number of packets in flight is less than the smaller of congestion window and the advertised receiver window. Clearly, to pace packet transmission and maintain a smooth sending rate, we disable this behavior for all data packets. However, we allow control packets (such as SYN, FIN, and RST) to be transmitted immediately to ensure proper operation of the underlying TCP/IP stack.

In addition, we implement a control timer function to periodically process the queue holding data packets. When the timer expires, the timer interval and the number of data packets to transfer during that interval are recalculated based on the designated data rate. Since a queued data packet cannot be sliced for transfer, the control interval is adjusted properly to take this into account in order to maintain the exact sending rate. Moreover, even though Linux supports delaying an event to a resolution of 1 μs, we chose the minimum timer interval to be 1 ms since small resolutions are achieved using CPU busy waiting and waste a lot of CPU cycles. This also helps to reduce the load on the system when thousands of connections are invoked concurrently.

3.3. Kernel tuning

The default Linux kernel network stack has many parameters that require tuning in order to support gigabit throughput for a wide range of RTTs. We increase the maximum size of both socket read and write buffers (`rmem_max`, `wmem_max`) from the default value of 131,071 bytes to 107,374,182 bytes, per-connection memory space defaults (`tcp_rmem`, `tcp_wmem`, `tcp_mem`) from (4096; 87,380; 174,760) bytes to (4096; 107,374,182; 107,374,182) bytes, size of the backlog queue (`netdev_max_backlog`) in the receive path from 300 to 10,000 slots, and that of the transmit queue in the forward path (`txqueuelen`) from 1000 to 10,000 slots. The size of transmit and receive ring buffers of the NIC is increased from 256 slots to the maximally possible value of 4,096 slots. We also disable the TCP segmentation offload and checksum verification features of NICs to support the new congestion header in our implementation. More information about the Linux network stack can be found in [16].

3.4. Congestion header

In explicit congestion control, congestion header is used to communicate congestion-related information and feedback between the end-hosts and routers. The congestion header size of XCP, JetMax, RCP, and PIQI-RCP in our implementation is 20, 32, 16, and 24 bytes, respectively. RCP has the smallest congestion header size while JetMax has the largest. Considering unidirectional data flow, i.e., data packets flow only in one direction and the other direction has only acknowledgement packets, the congestion header size can be further reduced to 12, 20, 8, and 13 bytes.

Congestion headers of all four protocols are shown in Fig. 2. All headers have the following common fields: (1) `Protocol` is the protocol number of the Transport Layer above them. For example, for TCP this value is 6; (2) `Length` is the size in bytes of the congestion header between the TCP and IP header; (3) `Version` is the protocol version of the AQM algorithm; and (4) `Unused` may be required later for possible protocol extensions. Its value should be set to zero.

For XCP, the congestion header is the same as suggested in [8]. `RTT` represents the end-host's current estimate of round-trip time measured in ms. Field `X` stores the inter-packet interval of a flow measured in ms. Variable `Delta` represents the desired throughput of a flow expressed in bytes per ms. Routers modify this field to represent the allocated change in throughput expressed in bytes per ms. Data receivers copy the value stored in the `Delta` field of received packets into the `Reverse` field and send it in acknowledgement packets. The total size of XCP's congestion header is 20 bytes.

JetMax's congestion header is elaborately discussed in [41]. Field `RT` stores the router ID of the currently known bottleneck router. The value of `RC` is incremented at every router encountered in a flow's path. It helps in calculating

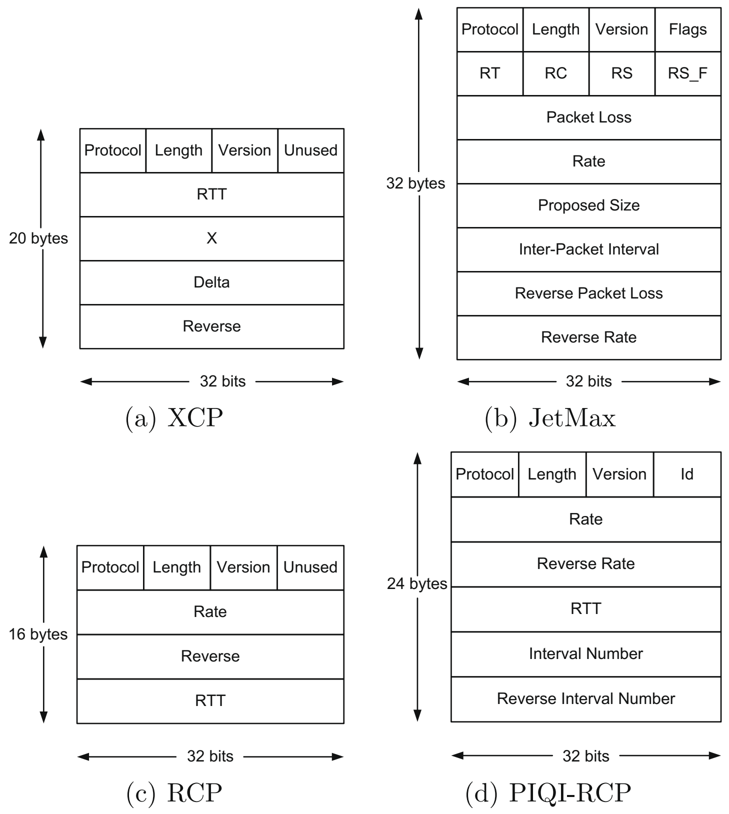

Fig. 2. Congestion header formats for XCP, JetMax, RCP, and PIQI-RCP.

the value of RT and RS. Field Packet Loss is used to store the virtual packet loss rate at routers as the packet passes through them. The router with the highest packet loss rate is considered to be the bottleneck router of each flow. Bottleneck switching takes place whenever a router has a packet loss rate that is higher than inside the current bottleneck router of a flow. Field RS stores the ID of the router in case a bottleneck switch is detected. Data receivers copy the value of RS in received packets into the RS_F field and send it in acknowledgement packets. Field Rate is modified by bottleneck routers and is used to assign sending rates to end-users. Field Proposed Size is used by flows to propose a new sending rate and request it to be approved by the routers in their path. Field Inter-Packet Interval stores the inter-packet interval for a flow expressed in ms. It assists router's estimation of the number of flows in the system. Data receivers copy the value of Packet Loss and Rate in the received packets to Reverse Packet Loss and Reverse Rate field respectively and send them in acknowledgement packets. The total size of JetMax's congestion header is 32 bytes.

In the case of RCP's congestion header, the Rate field is modified by routers to assign sending rates to flows. Field RTT represents the end-host's current estimate of the RTT measured in msec. Data receivers copy the value stored in the Rate field of the received packets into the Reverse field and send it in acknowledgement packets. The total size of RCP's congestion header is 16 bytes.

PIQI-RCP's congestion header is similar to that of RCP except for the three additional fields Interval Number, Reverse Interval Number, and ID. Field Interval Number is assigned by routers to indicate their control interval sequence numbers corresponding to the assigned rate feedback value. This field is copied into the Reverse Interval Number field while sending the acknowledgement packets. Variable Interval Number helps the end-host to identify the uniqueness of received feedback and invoke the control algorithm once per router's control interval. Field ID is the ID of the bottleneck router. The total size of PIQI-RCP's congestion header is 24 bytes.

3.5. Router

Routers in explicit congestion control algorithms need to provide end-hosts with accurate feedback of the actual congestion level. They gather information of the input traffic rate and queue size using some per-packet processing and then apply this information into a control equation at regular intervals to generate feedback, which is injected into every packet's congestion header and further fed back to the sender. This functionality has been implemented using Qdisc in prior work [37]; however, to design a more flexible framework, we take advantage of netfilter [28], which is a powerful packet filtering framework in the Linux network stack commonly used to develop firewall software such as Iptables. Using netfilter, custom user-defined functions known as "hooks" can be invoked at five different places in the IP and Route modules of the network stack.

Specifically, as illustrated in Fig. 3, before each incoming packet into the IP layer enters the Route module, it can be

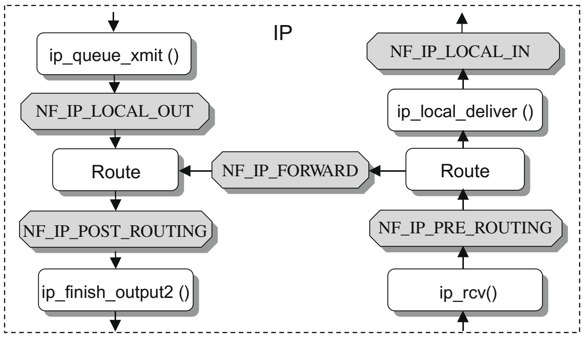

Fig. 3. Illustration of netfilter hooks in Linux TCP/IP stack.

intercepted by hook function NF_IP_PRE_ROUTING. After being processed by the Route module, packets to be delivered to the local host can be captured by NF_IP_LOCAL_IN and those to be forwarded to another host can be intercepted by NF_IP_FORWARD. Furthermore, outgoing packets from the local host will pass through NF_IP_LOCAL_OUT before being processed by the Route module. Finally, all outgoing (both locally-generated and transit) packets can be intercepted by NF_IP_POST_ROUTING. Thus, one can implement operations at one or more of these five interception points to realize desired AQM functionalities. We refer interested readers to [28] for more information of netfilter and its hook functions.

In our implementation, we develop AQM modules of RCP, PIQI-RCP, XCP, and JetMax in the hook function NF_IP_POST_ROUTING, at which point all outgoing packets from the local machine or incoming packets from other network interfaces that need to be forwarded are processed. We assign the hook function the lowest priority so that it is invoked only after all the kernel routines for processing the packet have been finished. The hook function collects information present in the congestion header of packets, updates the module's data structure with this information, and inserts feedback into the packets. A timer function is used to invoke the AQM-router's control equation at regular intervals T and generate the feedback information that is inserted into the packets during the subsequent interval. The duration of this interval is 10 ms for RCP and PIQI-RCP, 100 ms for JetMax, and the average RTT of all transit flows for XCP.

4. Experimental results

Using the implementation described above, we next provide an experimental examination of RCP, PIQI-RCP, XCP, and JetMax. We first set gain parameters to those suggested by the authors of each protocol: $\alpha = 0.4$ and $\beta = 0.226$ for XCP [19], $\tau = 0.6$ for JetMax [41], $\alpha = 0.1$ and $\beta = 1$ for RCP [6], and $\tau_1 = 0.01$, $\tau_2 = 0.1$, $\alpha = 0.5$, and $\gamma = 0.95$ for PIQI-RCP [17]. We then verify our implementation by replicating ns2 plots shown in the original papers [6,17,19,41]. This is a strong indication that explicit congestion control schemes, which rely on *accurate* router-generated feedback, may indeed be a viable solution to future high-speed heterogeneous networks and their theoretical properties (e.g.,

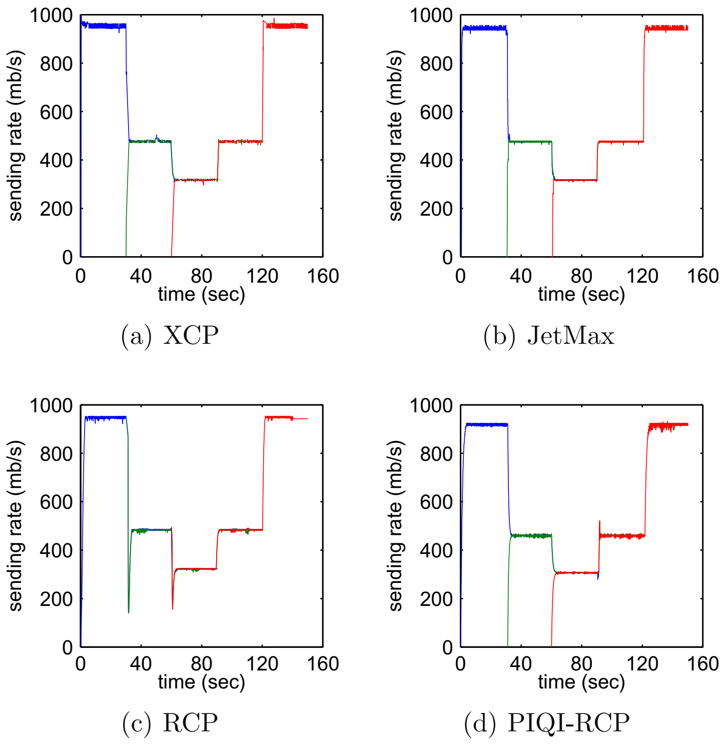

Fig. 4. Performance of three flows sharing a single-bottleneck link of capacity 1 gb/s and RTT 50 ms.

stability and fairness) can be realized with simple, yet highly efficient, implementations. In this paper, we do not include all those results, but focus on several representative scenarios that demonstrate *key* properties of these protocols.

In all experiments discussed below, we report the achievable throughput at the IP layer using packet size of 1500 bytes. Furthermore, all experiments are performed in Emulab [7] using Dell PowerEdge 2850 servers with 3.0 GHz 64-bit Xeon processors, 2 GB of RAM, and multiple gigabit network cards.

4.1. Single-bottleneck topology

We first examine the performance of these protocols in high-speed networks with a single-bottleneck link. Consider a dumb-bell topology where three flows pass through a single-bottleneck link of capacity[2] 1 gb/s and round-trip propagation delay 50 ms. Each flow is connected to the bottleneck link through a different access link of capacity 1 gb/s and negligible propagation delay. These flows start with a 30-s lag and each lasts for 90 s. Dynamics of the actual sending rates[3] of these protocols are illustrated in Fig. 4, in which all methods are able to maintain high sending rates. In all experiments presented in the paper, there is no packet loss and the sending rate is equal to the arrival rate at the receiver.

We also monitor queuing dynamics inside the bottleneck router for the above experiment and find that XCP and JetMax are successful in controlling their queue length at low levels during the entire experiment. However, RCP experiences significant queue buildup (up to 9415 packets) whenever a new flow joins the system. This phenomenon is attributable to the fact that when a flow starts, the bottleneck router cannot tell whether it is a new flow, which makes it assign the old fair rate to this flow. This causes the input traffic rate to overshoot the bottleneck link capacity and overflow the queue.

In contrast to RCP, PIQI-RCP is able to maintain low queues since new flows entering the system start with a small sending rate that increases gradually. This lends the router enough time to converge to a new steady-state feedback value without significantly overflowing the queue. As shown in the figure, PIQI-RCP has a slightly smaller throughput as compared to other methods since the router control algorithm operates on the virtual link capacity γC with $\gamma = 0.95$.

4.2. RTT unfairness

Unlike TCP Reno and many other end-to-end high-speed TCP variants [36], explicit congestion control methods do not suffer from RTT unfairness. Flows bottlenecked at a common router share equal rates irrespective of their RTTs. In this experiment, we use a dumb-bell topology where the bottleneck link has a capacity of 1 gb/s and

[2] Though the link capacity is 1 gb/s, the achievable IP layer throughput is 970 mb/s.

[3] The sending rate is averaged every two RTTs for JetMax, RCP, and PIQI-RCP, and is approximated by the ratio between the congestion window and smoothed RTT estimation (i.e., `cwnd/srtt`) for XCP.

propagation delay of 15 ms. The access link of flow x_1 connecting to the bottleneck router is 1 gb/s and has a 95-ms delay, while the access link of flow x_2 connecting to the bottleneck router has capacity of 1 gb/s and negligible delay. Hence the RTT of flow x_1 is 220 ms and that of flow x_2 is 30 ms, i.e., they differ by a factor of 7.

Fig. 5 illustrates dynamics of the two flows. First consider the scenario right after flow x_1 starts at $t = 0$. In the case of XCP and JetMax, x_1 achieves full link utilization almost instantaneously. In contrast, for RCP it takes x_1 nearly 10 s to saturate the link. This is because when flow x_1 initially joins the system, the computed rate at the router is very low. As the flow starts sending traffic, the control algorithm computes a new rate. It takes a number of iterations or control cycles before the router's control equation gives a rate close to the link capacity. Then, flow x_2 joins the system at $t = 30$ s. Again, XCP and JetMax converge to fairness almost immediately, while RCP spends over 5 s on clearing the built-up queue and giving both flows their fair share. As seen from part (d) of the figure, PIQI-RCP exhibits better convergence properties than RCP. In the case of PIQI-RCP, not only does x_1 saturate the virtual link capacity faster, but also does the system converge to its steady state almost instantaneously after x_2 joins the network.

XCP, being a window-based protocol, emits packets into the network in bursts. To support high throughput, flows with larger RTT have to maintain a larger congestion window. Because of these two reasons, flows with small RTT experience high variance in queuing delay. This can be easily seen in Fig. 5a. Flow x_2, with small RTT entering the system at $t = 30$ s, experiences small oscillations in its sending rate when co-existing with flow x_1.

4.3. Max–min fairness in XCP

Recall that JetMax, RCP, and PIQI-RCP all achieve provable max–min fairness in the steady state. In contrast, as demonstrated in [26], XCP is unable to guarantee max–min fairness and may result in arbitrarily unfair stationary resource allocation in certain topologies. We next verify this by considering the topology shown in Fig. 6a, which is composed of two links l_1 and l_2 and n^2 flows where n is a given constant. Flow x_1 passes only through link l_1 and the other $n^2 - 1$ flows $x_2 - x_{n^2}$ traverse both links. In addition, link capacities are set to $C_1 = 155$ and $C_2 = C_1(n-1)/n$ mb/s. For max–min fairness, the $n^2 - 1$ long flows should be congested at link l_2 with stationary sending rates $x_2^* = 155/[n(n+1)]$ mb/s and flow x_1 should converge its rate to $x_1^* = 155/n$ mb/s.

To examine whether XCP achieves max–min fairness in this topology, we plot in Fig. 6b ratios $\tilde{x}_1/x_1^*$ and $\hat{x}_1/x_1^*$ for different values of n, where $\tilde{x}_1$ is the sending rate of the flow predicted by the model developed in [26] and $\hat{x}_1$ is the actual sending rate measured in our experiment. These ratios indicate how close the system is to max–min fairness, i.e., the closer the ratios are to 1, the more max–min fair the system is. As shown in the figure, the system indeed departs from the max–min fair state when the number of flows increases. It is also clear that our experi-

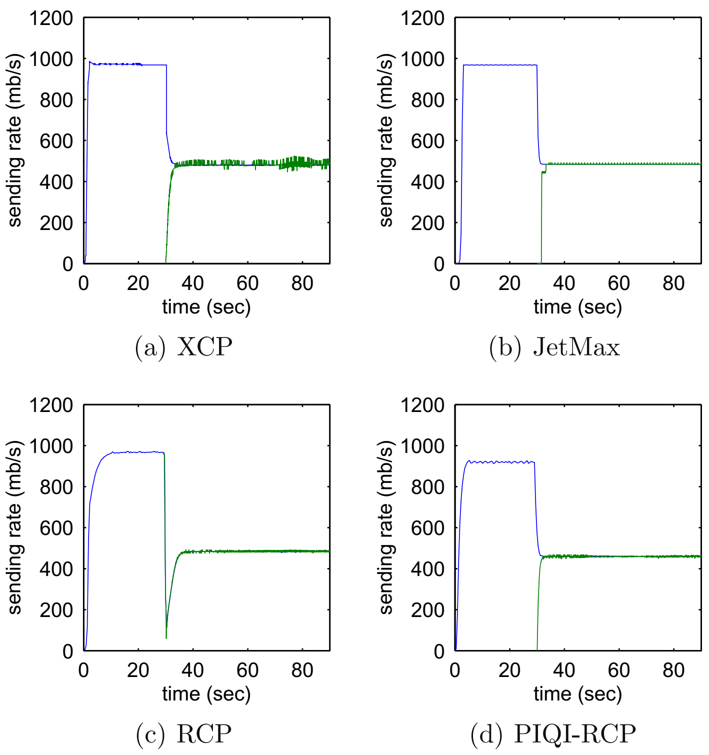

Fig. 5. Performance in the case of flows with heterogeneous RTTs.

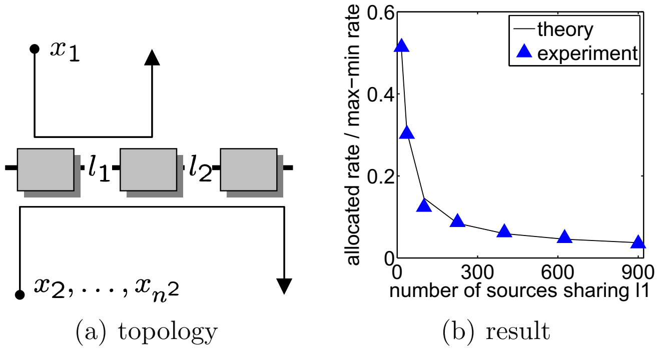

Fig. 6. Experimental verification of XCP's max–min fairness issue identified in [26].

mental results exactly match theoretical values predicated by [26].

4.4. CPU usage at routers

Due to per-packet processing of congestion headers of incoming packets, computation of feedback for every control interval, and stamping feedback on every outgoing packet, it may be believed that explicit congestion control would involve significant computational overhead, especially in high-capacity links, that can undermine their deployment. However, our experiments show that the overhead involved in these computations is virtually insignificant. Specifically, for a gigabit transfer from one sender to a receiver through a router, the average load on the router is around 30% for all the protocols such as TCP, XCP, JetMax, RCP, and PIQI-RCP. Hence, for 70% of the time, the router is idle. Out of the 30% load, most CPU time is utilized in the handling of IRQs (Hardware Interrupt Requests) and Software Interrupts. This indicates that the computational overhead involved is relatively small as compared to processing the interrupts invoked to enqueue and dequeue packets. This leads us to believe that explicit congestion control can be efficiently incorporated into the fast path of commercial routers and indeed offers a viable solution to future high-speed heterogeneous Internet.

4.5. Performance with mice traffic

In this section, we study the performance of explicit-feedback methods when the traffic is a combination of short- and long-lived flows. The test setup in this case is a dumb-bell topology with two senders and one receiver. We start the long flow at $t = 0$ from one of the two sender machines and generate the mice traffic (i.e., short-lived flows) at $t = 30$ s from the other machine. Both sets of flows pass through a common bottleneck link with capacity 1 gb/s and round-trip propagation delay 50 ms. The pattern of mice traffic's connection arrivals follows a Poisson process with mean inter-arrival time of 0.2 s and Pareto distributed traffic size with shape parameter 1.4 and mean of 100 packets. We note that as demonstrated in [10,29], Poisson arrival can precisely model the arrival processes of certain types of TCP connections (e.g., remote-login and file transfer), but is not suitable for many other connection arrivals. However, as explained below, results presented in this section do not depend on the arrival pattern of mice traffic and thus will hold for other traffic arrival processes.

As shown in Fig. 7a, in the case of XCP, the link utilization for the first 30 s is close to 100% when only the long flow is present in the system. But when mice traffic is started, the link utilization drops. The input traffic at the router fluctuates between 800 and 970 mb/s. The loss in link utilization is caused by a part of the feedback sent by the router to short flows not being utilized as they exit from the system after transferring a small number of packets. The system still operates at high utilization since XCP is conservative in giving bandwidth to new flows entering the system.

In the case of RCP, the link utilization remains very high with the input traffic rate occasionally overshooting link capacity by huge margins. This is evident from Fig. 7c. For the first 30 s, the input traffic at the router is stable around the link capacity. Fluctuations arise in the system after mice traffic arrives into the system at $t = 30$ s. New flows entering the system are given the rate calculated in the last control interval. As a result, when many flows enter the system simultaneously, the input traffic rate exceeds link capacity and causes high queue levels. This makes the router reduce the rate in the next control interval in order to drain the queue, which temporarily reduces link utilization. This behavior can be clearly seen at $t = 55$ s. Hence, in the case of RCP, large buffer size is recommended at routers to absorb the sudden rise in input traffic rate and prevent large amounts of packet loss.

As shown in Fig. 7d, PIQI-RCP also achieves high link utilization in this scenario. Throughout the experiment, the input traffic is very close to the set virtual link capacity without significant overshoots suffered by RCP. This is primarily because new flows entering the system start with a small sending rate that increases gradually upon receiving feedback from the router.

In contrast to other methods, JetMax behaves very differently in the presence of mice traffic. As seen in Fig. 7b, up to $t = 30$ s, JetMax operates at high utilization levels when there is only one long flow in the system. However, when mice traffic is started, the router perceives that a lot of flows have entered the system, the reason being the estimation of the number of flows N_l in the system is independent of the input traffic and depends only upon

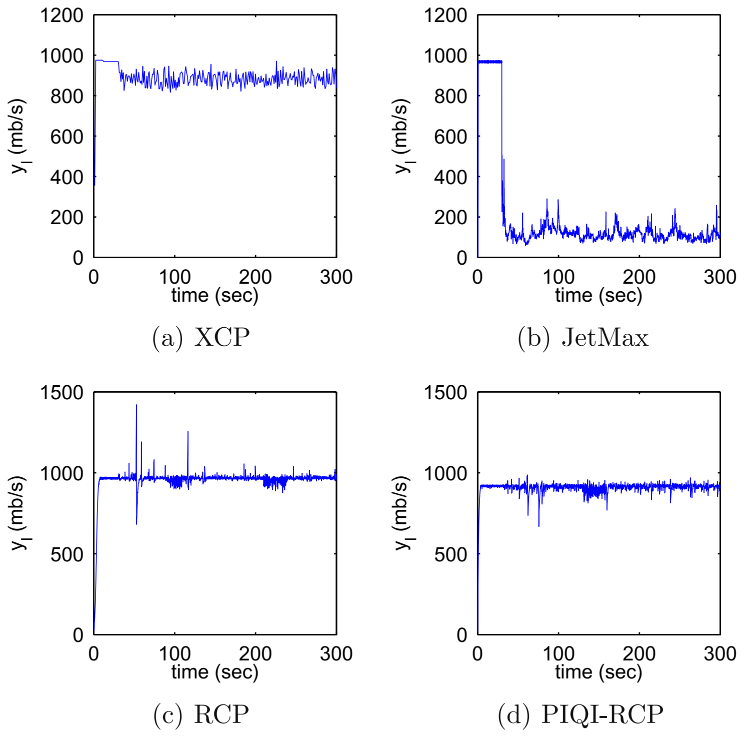

Fig. 7. Performance in the presence of background mice traffic generated in the system at $t = 30$ s.

the sum of inter-packet transmission delay set in the congestion header by the flows. In the presence of mice traffic, the system is unable to differentiate between short and long flows and feeds back an equal rate to all of them. This causes the long flow to significantly decrease its sending rate, i.e., the long flow gets penalized in the presence of short flows. For the current setup, the link utilization fluctuates between 10-20% and most flows last less than one RTT and therefore are unable to fully utilize the bandwidth allocated by the router.

4.6. Abrupt change in traffic demand

In this experiment, we examine the performance of the system with abrupt increase or decrease in traffic demand. We use a dumb-bell topology with two machines on one side of the bottleneck acting as senders and one machine on the other side acting as the receiver. All access links have capacity 1 gb/s, while the bottleneck link has capacity 100 mb/s and round-trip propagation delay 50 ms. At $t = 0$, one long flow is started for a 120 s. At $t = 30$ s another 10 flows abruptly enter the system. These 10 flows continue to remain in the system until $t = 113$ s, when they all suddenly exit. The performance of these protocols in this scenario is shown in Fig. 8.

From the figures, it can be clearly inferred that XCP and JetMax are robust in the face of sudden increase or decrease in traffic demand. Link utilization drops momentarily when a large number of flows join or leave the system simultaneously, but the system converges very fast to its steady state. RCP performs the worst in this case. At time $t = 30$ s, the average input traffic overshoots to around 300 mb/s and the queue size jumps to around 11,000 packets as shown in Fig. 9a. This is because at this time, the router's control algorithm has the per flow rate computed to be 100 mb/s, which is given to all incoming flows. When flows start sending at this rate, the queue at the router significantly builds up. Next, with the rise in both the input traffic rate and queue size, the router's control equation computes a very low rate. This rate, when assigned to the flows, makes them drastically reduce their sending rate and hence cause the drop in link utilization. The system remains in this transient state (i.e., both overshoot and undershoot) for about 7 s (i.e., 140×50 ms $= 140$ RTTs) before reaching its steady state. In the case of PIQI-RCP, the overshoot in average input traffic at $t = 30$ s, as compared to RCP, is significantly lower. As can be seen from the figure, it increases to around 128 mb/s but the system quickly converges to its steady state. The excessive traffic, for a short period, gets absorbed in the network device queues without increasing the IP layer queue.

4.7. Multiple-bottleneck topology

We next examine the performance of these protocols in a parking-lot topology shown in Fig. 9b, which is composed of two bottleneck links (l_1, l_2) and three flows (x_1, x_2, x_3). Capacity of these two links are $C_1 = 970$ and $C_2 = 800$ mb/s, and the round-trip propagation delay of each link is 50 ms. Flow x_1 passes through both links, but flows x_2 and x_3 only utilize l_1 and l_2. Flow x_1 starts first and converges its rate to the capacity of l_2, i.e., 800 mb/s. When x_2 joins 30 s later, x_1 switches its bottleneck to l_1 and both flows converge to an even share of $C_1/2 = 485$ mb/s. As x_3

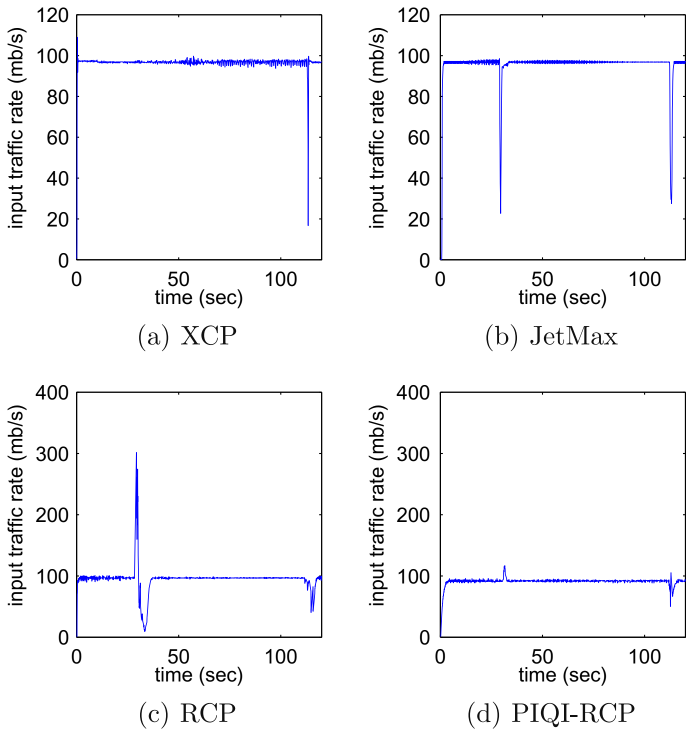

Fig. 8. Performance under abrupt change in traffic demand.

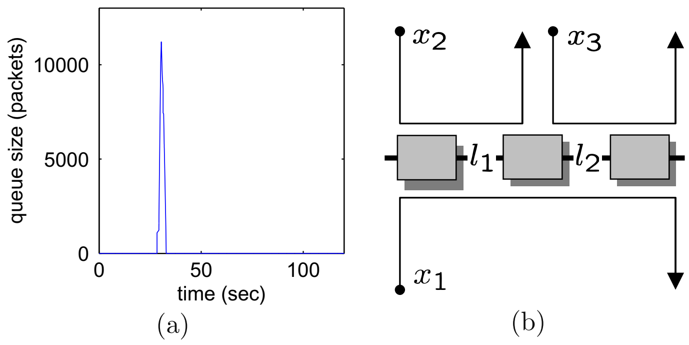

Fig. 9. (a) Queuing dynamics in RCP under abrupt change in traffic demand and (b) multiple-bottleneck topology.

starts at time $t = 60$ s, x_1 changes its bottleneck back to l_2 and converges its sending rate together with x_3 to $C_2/2 = 400$ mb/s. Flow x_2 then utilizes the remaining bandwidth on link l_1, i.e., 570 mb/s. Finally, when x_1 terminates at $t = 90$ s, flows x_2 and x_3 change their sending rates to the capacity of each link. As demonstrated in Fig. 10a–d, all methods are stable and max–min fair in this case with their dynamics following the theoretical understanding. In the case of PIQI-RCP, the sending rate of flows $x_1 - x_3$ are scaled by $\gamma = 0.95$ since the router controller operates on the virtual link capacity γC with $\gamma = 0.95$.

4.8. Summary of results

From the above experimental results, we find all these protocols to be scalable with the increase in link capacity and round-trip propagation delay. Also, in the steady state, they admit low queuing delay and almost zero packet loss rate. XCP can maintain high link utilization and has low buffering requirements, but does not achieve max–min fairness in certain topologies and has the highest number of per-packet computations (6 additions and 3 multiplications) at routers. JetMax performs well in almost all scenarios using long flows, has the least buffering requirement, and requires the smallest number of per-packet computations (3 additions per packet for responsive flows, 2 additions per-packet for unresponsive flows, and zero multiplications). However, JetMax loses link utilization in the presence of mice flows. RCP is able to maintain high link utilization in most traffic scenarios and does a reasonable amount of per-packet computations (2 additions and 2 multiplications). However, it has a very high buffering

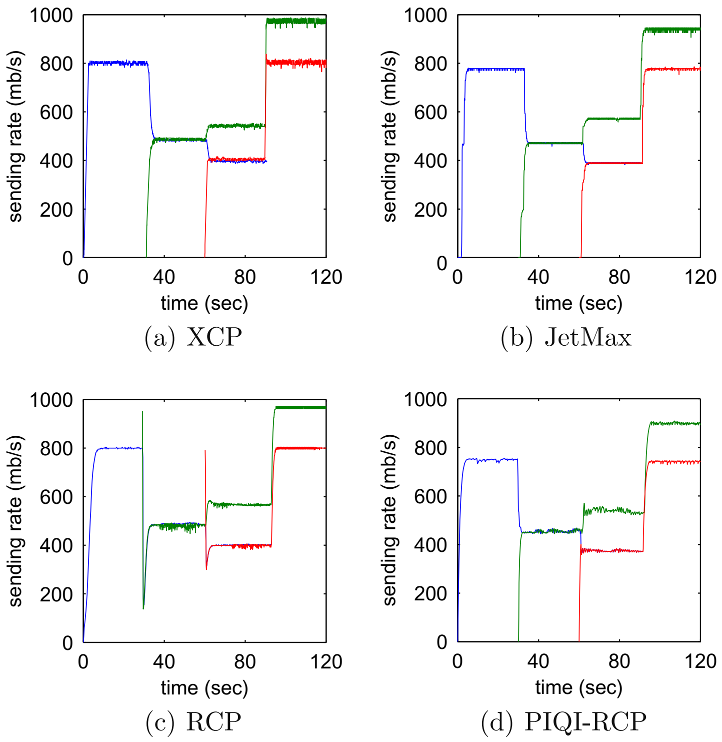

Fig. 10. Performance in a multiple-bottleneck link topology.

requirement to avoid packet loss and shows significant oscillations in the sending rate with abrupt increase or decrease in traffic demand. PIQI-RCP retains most of the strengths of RCP, has significantly lower buffering needs, and is more robust to abrupt surge in traffic demand.

5. Conclusion

In this work, we designed and implemented a novel *unified* Linux framework for developing explicit congestion control protocols. This framework allows for low-overhead implementations of both window- and rate-based congestion control algorithms without any interference with Linux's TCP/IP stack or the application layer. This significantly facilitates testing and deployment of explicit congestion control and can be easily extended to any new protocols. Using this implementation framework, we evaluated several existing methods in gigabit networks. Besides verifying our implementation to be in conformance with original `ns2` simulations, we also discovered several novel drawbacks of RCP (i.e., slow convergence to equilibrium and significant overshoot in the presence of abrupt traffic load changes) and JetMax (i.e., low link utilization under mice traffic). Our results lead us to conclude that PIQI-RCP performs the best among the studied methods given the scenarios considered in this work. In our future work, we would like to extend our study considering different traffic patterns, evaluate these protocols in networks with higher link capacities as they become accessible to us for experimentation, study their behavior in the presence of wireless links with time-varying capacity, and propose improvements to these protocols.

References

[1] J.S. Ahn, P.B. Danzig, Z. Liu, L. Yan, Evaluation of TCP vegas: emulation and experiment, in: Proceedings of the ACM SIGCOMM, August 1995, pp. 185–195.
[2] M. Allman, V. Paxson, W. Stevens, TCP congestion control, IETF RFC 2581, April 1999.
[3] A.A. Awadallah, C. Rai, TCP-BFA: buffer fill avoidance, in: Proceedings of the IFIP HPN, September 1998, pp. 575–594.
[4] S. Bhandarkar, S. Jain, A.L.N. Reddy, Improving TCP performance in high bandwidth high RTT links using layered congestion control, in: Proceedings of the PFLDnet, February 2005.
[5] L. Brakmo, S. O'Malley, L. Peterson, TCP vegas: new techniques for congestion detection and avoidance, in: Proceedings of the ACM SIGCOMM, August 1994, pp. 24–35.
[6] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, N. McKeown, Processor sharing flows in the internet, in: Proceedings of the IEEE IWQoS, June 2005.
[7] Emulab. [Online] Available: <http://www.emulab.net/> (accessed 01.08.2008).
[8] A. Falk, Y. Pryadkin, D. Katabi, Specification for the Explicit Control Protocol (XCP), IETF Internet Draft, July 2007.
[9] Fedora Core. [Online] Available: <http://fedora.redhat.com/> (accessed 01.08.2008.
[10] A. Feldmann, A.C. Gilbert, W. Willinger, Data networks as cascades: investigating the multifractal nature of internet WAN traffic, in: Proceedings of the ACM SIGCOMM, August 1998, pp. 42–55.
[11] FIND. [Online] Available: <http://find.isi.edu/> (accessed 01.08.2008.
[12] S. Floyd, High-speed TCP for large congestion windows, IETF RFC 3649, December 2003.
[13] S. Floyd, T. Henderson, The NewReno modification to TCP's fast recovery algorithm, IETF RFC 2582, April. 1999.
[14] GENI. [Online] Available: <http://www.geni.net/> (accessed 01.08. 2008.
[15] M. Gerla, M.Y. Sanadidi, R. Wang, A. Zanella, C. Casetti, S. Mascolo, TCP westwood: congestion window control using bandwidth

estimation, in: Proceedings of the IEEE GLOBECOM, November 2001, pp. 1698–1702.
[16] T. Herbert, The Linux TCP/IP Stack: Networking for Embedded Systems, Charles River Media, 2004.
[17] S. Jain, D. Loguinov, PIQI-RCP: design and analysis of a rate-based explicit congestion control, in: Proceedings of the IEEE IWQoS, June 2007.
[18] S.-R. Kang, D. Loguinov, IMR-pathload: robust available bandwidth estimation under end-host interrupt delay, in: Proceedings of the PAM, April 2008, pp. 172–181.
[19] 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.
[20] 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.
[21] T. Kelly, Scalable TCP: improving performance in high-speed wide area networks, Comput. Commun. Rev. 33 (2) (2003) 83–91.
[22] R. King, R. Riedi, R. Baraniuk, Evaluating and improving TCP-Africa: an adaptive and fair rapid increase rule for scalable TCP, in: Proceedings of the PFLDnet, February 2005.
[23] 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.
[24] A. Kuzmanovic, E. Knightly, TCP-LP: a distributed algorithm for low priority data transfer, in: Proceedings of the IEEE INFOCOM, April 2003, pp. 1691–1701.
[25] D. Leith, R. Shorten, H-TCP protocol for high-speed long distance networks, in: Proceedings of the PFLDnet, February 2004.
[26] 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.
[27] I.J. Nagrath, M. Gopal, Control Systems Engineering, John Wiley & Sons, 2004.
[28] Netfilter. [Online] Available: <http://www.netfilter.org/> (accessed 01.08.2008.
[29] V. Paxson, S. Floyd, Wide area traffic: the failure of poisson modeling, IEEE/ACM Trans. Netw. 3 (3) (1995) 226–244.
[30] I. Rhee, L. Xu, CUBIC: a new TCP-friendly high-speed TCP variant, in: Proceedings of the PFLDnet, February 2005.
[31] Network Simulator. [Online] Available: <http://www.isi.edu/nsnam/ns/> (accessed on 01.08.2008.
[32] H. Sivakumar, R.L. Grossman, M. Mazzucco, Y. Pan, Q. Zhang, Simple available bandwidth utilization library for high-speed wide area networks, J. Supercomput. (2003).
[33] M. Suchara, R. Witt, B. Wydrowski, TCP MaxNet–implementation and experiments on the WAN in lab, in: Proceedings of the IEEE ICON, November 2005, pp. 901–906.
[34] B.P. Wydrowski, L.L.H. Andrew, I.M.Y. Mareels, MaxNet: faster flow control convergence, Networking 3042 (2004) 588–599.
[35] B.P. Wydrowski, M. Zukerman, MaxNet: a congestion control architecture for maxmin fairness, IEEE Commun. Lett. 6 (11) (2002) 588–599.
[36] L. Xu, K. Harfoush, I. Rhee, Binary increase congestion control (BIC) for fast, long distance networks, in: Proceedings of the IEEE INFOCOM, March 2004, pp. 2514–2524.
[37] 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.
[38] Y. Zhang, S.-R. Kang, D. Loguinov, Delay-independent stability and performance of distributed congestion control, IEEE/ACM Trans. Netw. 15 (6) (2007) 838–851.
[39] 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.
[40] Y. Zhang, D. Leonard, D. Loguinov, JetMax: scalable maxmin congestion control for high-speed heterogeneous networks, Elsevier Comput. Netw. 52 (6) (2008) 1193–1219.
[41] Y. Zhang, D. Leonard, D. Loguinov, JetMax: scalable max–min congestion control for high-speed heterogeneous networks, in: Proceedings of the IEEE INFOCOM, April 2006, pp. 1–13.

Yueping Zhang received the B.S. degree in computer science from Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001 and the Ph.D. degree in computer engineering from Texas A&M University, College Station, USA, in 2008. He is currently a research staff member at NEC Laboratories America, Inc. His research interests include congestion control, delayed stability analysis, active queue management (AQM), router buffer sizing, and peer-to-peer networks.

Saurabh Jain received the B.S. degree in electronics and communication engineering from Indian Institutes of Technology, Roorkee, India, in 2003 and the M.S. degree in electrical and computer engineering from Texas A&M University, College Station, TX, in 2007. He is currently with Cisco Systems, Inc. His research interests include Internet congestion control and performance analysis.

Dmitri Loguinov received the B.S. degree (with honors) in computer science from Moscow State University, Moscow, Russia, in 1995 and the Ph.D. degree in computer science from the City University of New York, New York, in 2002. Since September 2007, he has been an Associate Professor of computer science with Texas A&M University, College Station. His research interests include peer-to-peer networks, Internet video streaming, congestion control, image and video coding, Internet traffic measurement and modeling.