One More Bit Is Enough

Yong Xia, RPI, xiay@alum.rpi.edu
Lakshminarayanan Subramanian, UCB, lakme@cs.berkeley.edu
Ion Stoica, UCB, istoica@cs.berkeley.edu
Shivkumar Kalyanaraman, RPI, shivkuma@ecse.rpi.edu

***Abstract*— Achieving efficient and fair bandwidth allocation while minimizing bottleneck queue length and congestion-induced packet loss rate for high Bandwidth-Delay Product (BDP) networks has long been a daunting challenge. The current end-to-end congestion control schemes including TCP and its many variants and the TCP+AQM/ECN proposals have significant limitations in achieving this goal. XCP, while achieving the goal, requires multiple bits (not available in the IP header) to exchange the congestion information between network and end-hosts. Adding these bits to the IP header is a non-trivial and lengthy process.**

In this paper, we design and implement a simple, low-complexity algorithm called Variable-structure congestion Control Protocol (VCP) that leverages only the existing two ECN bits for network congestion feedback and yet achieves comparable performance as XCP, i.e., exponential convergence onto high utilization, low persistent queue length, negligible packet loss rate and reasonable fairness. However, VCP has significantly slower fairness convergence speed in comparison to XCP. Extensive ns2 simulations validate the performance of VCP for a wide range of high BDP scenarios. Additionally, VCP can also be extended to provide bandwidth differentiation.

I. Introduction

The classical Additive-Increase/Multiplicative-Decrease (AIMD) algorithm [9] used in TCP congestion control [19] is well-known to be ill-suited for high Bandwidth-Delay Product (BDP) networks. With rapid advances in the deployment of very high bandwidth links in the Internet, the need for a viable replacement for TCP in such environments has become increasingly paramount.

Several research efforts have proposed different approaches for this problem, each with their own strengths and limitations. These can be broadly classified into two categories: *end-to-end* and *network feedback* based. We contend that pure end-to-end congestion control schemes like HighSpeed TCP [14], STCP [30], FAST TCP [24] and BIC TCP [49], although being attractive short-term improvements, are not suitable in the long term. The primary reason is that, in high BDP networks, using loss and/or delay as the only congestion signal(s) has fundamental limitations in achieving high utilization and fairness while maintaining a low bottleneck queue length and minimizing congestion-induced packet drop rate. HighSpeed TCP outlines the limitations of loss-based approaches in high bandwidth optical links with very low bit-error rates [14]. On the other hand, delay-based approaches like FAST TCP are sensitive to delay variations, which is a very common phenomenon in the Internet. For example, in a recent Internet experiment, FAST TCP throughput is 4$\sim$8 times less than the other tested stacks due to reverse path queueing delay [6].

The limitations of pure end-to-end schemes have motivated the use of explicit network feedback for congestion control. XCP [27], a promising proposal for high BDP networks, is an example of an *explicit rate* feedback scheme where the bandwidth allocation functionality is primarily placed inside the network. (ATM ABR mechanisms, e.g., [8][25][21][26], also belong to this class.) On the other hand, existing *congestion notification* feedback schemes like TCP+AQM/ECN proposals [15][2][34][45], in which end hosts adjust their sending rates based on congestion notification from the network, are known to not scale to high BDP scenarios [27].

Contrary to conventional wisdom, this paper demonstrates the existence of a simple congestion notification based scheme, namely Variable-structure congestion Control Protocol (VCP), that can approximate XCP's superior performance in high BDP networks with only two bits of feedback from network. VCP introduces two simple modifications to the TCP+AQM/ECN approach. First, VCP uses a load factor based mechanism to encode congestion information in the two ECN bits available in the IP header. Second, guided by the two bits of congestion feedback, VCP uses a *sliding-mode* congestion control algorithm as described below.

The basic idea behind the design of VCP is as follows. Network routers classify the level of congestion into three regions (i.e., low-load, high-load and overload), which can be encoded in the two ECN bits. Based on this encoded load factor feedback, end hosts switch their control algorithm between multiplicative increase (MI), additive increase (AI) and multiplicative decrease (MD) to match those three network load regions, respectively. By using MI in low-load region, flows can exponentially ramp up their bandwidth to improve network utilization. Once high utilization is attained, AIMD provides long-term fairness among competing flows.

We implement VCP in ns2 [41] and use extensive packet-level simulations to evaluate its performance and compare it with XCP. These simulations cover a wide range of network scenarios, particularly high BDP networks, and demonstrate that it is possible to design an end-host based congestion control with a two-bit ECN feedback to obtain most benefits of XCP, i.e., exponential convergence to high utilization, low persistent queue, negligible packet drop rate and reasonable fairness. One disadvantage of VCP when compared to XCP is a significantly slower fairness convergence speed.

VCP is attractive due to two reasons: (1) It does not require any modifications to the IP header. Since the ECN proposal has standardized two bits for congestion notification, VCP can just reuse these two bits. (2) It is a simple protocol with low algorithmic complexity. The complexity of VCP's end host algorithm is similar to that of TCP and part of its AIMD component is already present in the existing TCP code. Its router algorithm is extremely simple, scalable and maintains no per-flow state. We believe that these benefits largely offset VCP's limitation of having a slow fairness convergence speed.

Is two bits indeed necessary? Fundamentally, to achieve high utilization and long-term fairness it is essential to distinguish between three load regions and at least two bits are necessary to encode these three regions. Section II has more discussions on this. Another natural question is what would be the improvement of using more than two bits to encode the load regions. While we do not answer this question in the paper, the relatively small performance gap between VCP and XCP in Section IV suggests that using more bits will only incrementally improve the performance.

The rest of the paper is organized as follows. Section II discusses high level design rationales and illustrates VCP with a simple example. Section III describes VCP building blocks and the complete protocol. Section IV presents ns2 simulation results and compares with XCP. Section V analyzes the stability of VCP, addresses concerns on the influence of mode switching on efficiency and fairness and discusses incremental deployment issues. We review related work in Section VI and present our conclusions in Section VII.

II. Foundations

In this section, we provide a high-level description of VCP and illustrate how it works. Before we describe VCP, we briefly outline the reasons why TCP+AQM does not scale to high BDP networks while XCP does. Based on this, we extract two basic guidelines that form the core of VCP's design.

A. Why XCP outperforms TCP and TCP+AQM?

Using loss as the only congestion feedback, TCP cannot scale to high BDP networks. As per the TCP throughput equation [42], a TCP flow needs to observe an abysmally low loss rate (i.e., $O(10^{-14})$) to obtain a steady state throughput of 10Gbps with an RTT of 100ms [14]. Two of the primary problems for TCP's low utilization are: (a) loss is a poor congestion signal; (b) using the AIMD algorithm. Packet loss may be caused by events other than congestion. Even if a loss indeed signals congestion, it is a *binary* indicator of congestion and conveys no information about the degree of congestion. Given that this congestion signal is imprecise, the increase policy needs to be conservative while the decrease policy needs to be aggressive [27]. In high BDP networks, every loss event forces a TCP flow to perform an MD which is followed by the slow convergence of AI algorithm to reach high utilization levels. Given that time for each individual AIMD epoch is proportional to the network bandwidth, TCP flows remain in low utilization regimes for prolonged periods of time thereby resulting poor link utilization. Using AQM/ECN in conjunction with TCP does not solve this problem either. While ECN feedback prevents congestion collapse, this feedback is also *binary*. The existing AQM schemes with one-bit ECN indicate congestion near 100% utilization and provide no information on the degree of congestion.

XCP offers a complete solution to the congestion control in high BDP network. It precisely measures the bottleneck spare bandwidth to detect and report congestion. It decouples efficiency control and fairness control *at the router*: MIMD is used to control the flow aggregate and converge exponentially fast to any available bandwidth, while AIMD is used to fairly allocate this bandwidth among competing flows. Consequently, XCP significantly outperforms TCP and TCP+AQM in high BDP networks. However, XCP needs multiple bits in the packet header to carry (a) bandwidth allocation information ($\Delta cwnd$) from network routers to end hosts, and (b) congestion window ($cwnd$) and RTT information (rtt) from the end-hosts to the network routers.

B. Design Guidelines

Based on the above description, we realize that it is essential to: (a) measure the degree of network utilization and match end host sending rates with the degree of utilization; (b) use different algorithms to control efficiency and fairness. A key question is to determine how to implement these mechanisms at *end hosts* with *minimal network assistance*. By network assistance, we refer to the AQM-style congestion control approach where routers provide feedback on the level of network congestion and end hosts perform the actual congestion control mechanism using this feedback. Designing an end-host based congestion control for high BDP networks which uses minimal network assistance is a challenging problem, especially since end hosts have to independently adjust their sending rates using only the feedback available from the network. Unlike XCP, an end-host based scheme does not have the luxury of interacting with network routers by signaling end-host congestion state and obtaining end-host specific feedback. To address this challenge, we use two basic design guidelines:

Guideline #1, Decouple efficiency control and fairness control in different bandwidth utilization regions.

Efficiency and fairness have different levels of relative importance under different situations. When bandwidth utilization is low, our goal is to improve efficiency more than fairness. However, when bandwidth utilization is sufficiently high, we accord higher priority to fairness than efficiency. By decoupling these two issues in different utilization regions, end hosts have only a single objective in each utilization region and thus need to apply the appropriate congestion response in each regime. For example, one such choice of a congestion response which we use in VCP is to perform MI in low utilization regimes for improving efficiency, and use AIMD in high utilization regions for fairness. The goal then is to switch between these two congestion responses depending on the current network utilization region.

Guideline #2, Use link load factor as the congestion signal.

XCP uses spare bandwidth, i.e., the difference between demand and capacity to measure the degree of congestion. In this paper we use load factor, i.e., the relative ratio of demand and capacity [21]. This selection satisfies the above requirement of differentiating utilization regions.

Load factor conveys less information than spare bandwidth; and this may fundamentally bound the performance of a load

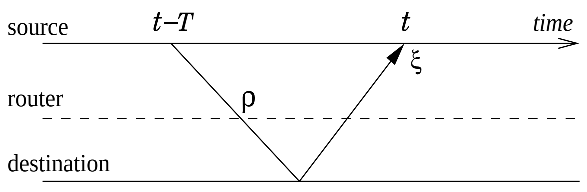


Fig. 1. A simplified VCP model. The source sending rate at time $t - T$ is used by the router to calculate a load factor ρ, which is echoed back from the destination to the source at time t. Then the source adjusts its MI parameter $\xi(t)$ based on the load factor $\rho(t)$.

factor based scheme. However, the advantage of using load-factor is that it is a *scale-free* property in terms of network capacity. It thus can be encoded using a small number of bits without much loss of information (since we are generally interested in the load factor range as opposed to precise feedback). While one can develop mechanisms to encode explicit rate feedbacks like XCP using a smaller number of bits, one would require at least $O(\log B)$ bits to be able to scale to a maximum network bandwidth B. However, a load factor can be encoded in as small as two bits which we show is sufficient to approximate XCP's performance.

Load factor, in comparison to binary congestion signals such as loss and one-bit ECN, can convey more information about the degree of network congestion. So we can expect that, together with a set of matching bandwidth-probing algorithms at end hosts, a load factor based scheme has the potential to perform much better than the loss-based or one-bit ECN-based congestion control algorithms.

C. VCP: A Simple Illustration

Now, we provide a simple illustration of how VCP works using a toy example of a bottleneck link shared by two competing flows. We present a detailed description of VCP in Section III. Periodically, the router measures a load factor for its output link (Figure 1). The load factor is echoed back to the source hosts via the destination acknowledgment (ACK). Depending on this feedback, the end-hosts apply different congestion responses. If the link signals significant underload, the end-hosts increase their sending rates using MI; if it signals marginal underload, they increase their sending rates using AI; otherwise, if it signals overload, they immediately cut their sending rates using MD. This is summarized by the following greatly simplified VCP pseudo code.

1) Each router periodically estimates a load factor, and encodes this load factor into the data packet headers. This information is then sent back by the receiver to the sender via ACK packets.

2) Based on the value of the load factor it receives, each source performs the following congestion control algorithms:

   2.1) For significant underload, perform MI;
   2.2) For marginal underload, perform AI;
   2.3) For overload, perform MD.

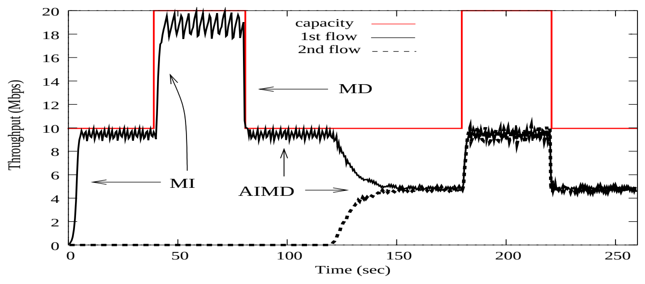


Fig. 2. The throughput dynamics of two flows of the same RTT (80ms). They share one bottleneck with the capacity bouncing between 10Mbps and 20Mbps. This simple example unveils VCP's potential to quickly track change in available bandwidth (with load factor-guided MIMD) and thereafter achieve fair bandwidth allocation (with AIMD).

We continue to describe more details of the above algorithm and explain why this simple algorithm works. Consider a single link used by one flow that has constant RTT value $rtt = T$ where the link queue has infinite buffer space. With only one flow, we have a single goal: to achieve efficiency for any link capacity (with load factor-guided MIMD according to the design guidelines). As shown in Figure 1, due to the round-trip delay, the load factor received by the *source* is:

$$\rho(t) = \frac{cwnd(t-T)}{cT}, \tag{1}$$

where $cwnd(t)$ is the source host congestion window size at time t, and c is the link capacity.

Since $\rho(t)$ represents the offered load (with delay), $1-\rho(t)$ measures the (normalized) spare bandwidth. Upon receiving a load factor, if the source updates its congestion window using MI:

$$cwnd(t+T) \;=\; cwnd(t) \times (\,1 + \xi(t)\,) \tag{2}$$

with

$$\xi(t) = \kappa \cdot \frac{1-\rho(t)}{\rho(t)} \tag{3}$$

where $\kappa > 0$, then it can achieve high efficiency since it exponentially tracks *any* available bandwidth. The stability of the above algorithm is determined by the choice of the value of κ. (Refer to Section V-B for the stability discussion.)

Now, assume that a new flow with the same RTT starts transmitting. Besides efficiency, now our other goal is to allocate bandwidth fairly (in the long run) between the two flows. Following the design guidelines, while in the low utilization region $\rho(t) < \rho_0$, we use the MIMD algorithm described above to quickly improve the efficiency. Next, we apply the standard AIMD algorithm with parameters α and β to achieve fairness in the high utilization region $\rho(t) \geq \rho_0$:

$$cwnd(t+T) \;=\; cwnd(t) + \alpha, \quad \forall \rho(t) \in [\rho_0, \rho_1) \tag{4}$$
$$cwnd(t+\delta t) \;=\; cwnd(t) \times \beta, \quad \forall \rho(t) \in [\rho_1, \infty) \tag{5}$$

where $\delta t \to 0+$ and $0 < \rho_0 \leq \rho_1 \leq 1$. (The setting of these parameters will be discussed in Section III.)

Figure 2 shows the throughput dynamics of two flows controlled by the above VCP algorithm. Clearly, bandwidth changes are quickly tracked by MIMD and, when a new flow arrives, the existing flow yields bandwidth to it with AIMD. Finally, an efficient and fair allocation is reached.

The Internet, however, is much more complex than this substantially simplified model: link capacity and router buffer size are dramatically heterogeneous; the number of flows is unknown and usually constantly changing; their RTTs may differ significantly; and so on. We need to generalize the above model to more realistic environments.

III. VCP: The Complete Protocol

In this section, we provide a complete description of VCP. To make VCP work under the Internet's significant heterogeneity in link capacity, end-to-end delay, router buffer size, and traffic characteristics, we need to address four key questions: (a) Given the fact that the traffic seen by a router is bursty, how do we compute a load factor that reliably estimates the link load condition at the *right* time scale? (b) Consider the particular case in which all flows have the same RTTs. How do we set the MI/AI/MD algorithm parameters to achieve efficiency and fairness, and at the same time maintain low persistent bottleneck queues and minimize congestion-induced packet losses? (c) When flows have different RTTs, one can imagine that different flow's MI/AI/MD mode sliding will be out of sync and the aggregate behavior will be unpredictable. Is it possible, and if yes how, to offset the impact of the RTT heterogeneity? (d) How can we provide equal (or differentiated) bandwidth sharing using window-based congestion control? Next, we describe VCP's four key building blocks that answer these questions and take us step-by-step through the design of a practical VCP algorithm.

A. Link Load Factor Measurement and Encoding

Due to the bursty nature of Internet traffic [37][44], an instantaneous load factor as in Equation (1) does not truly represent the real load condition. We need to measure a load factor over an appropriate time interval. Network congestion may persist over different time scales. Since we are concerned with those congestion events that last longer than one RTT, we choose a measurement interval t_ρ larger than one RTT of a majority of flows. Every $t_\rho = 200$ms, [1] each router estimates the load factor for each of its output link l with:

$$\rho_l = \frac{\lambda_l + \kappa_q \cdot \widetilde{q_l}}{\kappa_l \cdot c_l \cdot t_\rho} \tag{6}$$

where $\kappa_q > 0$ and $0 < \kappa_l \leq 1$. (We use $\kappa_q = 0.5$ and $\kappa_l = 0.98$.) Here λ_l is the amount of input traffic during the period t_ρ, $\widetilde{q_l}$ is the persistent queue length during this period, κ_q controls how fast the persistent queue drains [2], κ_l is the target utilization [34] and c_l is the link capacity. The input traffic λ_l is measured using a simple packet counter. The persistent queue $\widetilde{q_l}$ is generated by low-pass filtering the instantaneous queue $q(t)$ using a timer-driven approach that measures $q(t)$ every t_q = 10ms $\ll t_\rho$. Alternatively, we can also use a data driven approach as used in RED [15] which makes a measurement upon very packet enqueue event.

[1]Internet measurement shows that 90-95% of flows have an RTT less than 500ms. For US links, 75-90% of flows' RTT is less than 200ms [23].

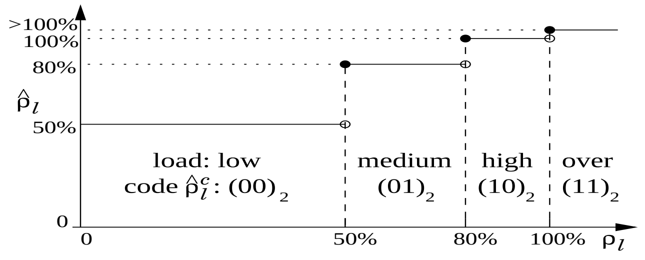


Fig. 3. The quantized load factor $\hat{\rho}_l$ at a link l is a non-decreasing function of the raw load factor ρ_l and can be represented by a two-bit code $\hat{\rho}_l^c$.

Given sufficient demand, our goal is to keep each link working around a state where $\rho_l = 100\%$ and κ_l is slightly less than one. If we are able to reach this state, we have $\lambda_l/t_\rho = \kappa_l\, c_l$. Substituting this in Equation (6), we get $\widetilde{q_l} = 0$. Consequently, in this state, the link capacity is almost fully utilized and there is no persistent queue in the router buffer.

Being a normalized measure, the load factor ρ_l can be encoded using a small number of bits with sufficient precision. Generally, given n bits, the whole load factor range $[0, \infty)$ can be partitioned into $2^n - 1$ underload segments covering $[0\%, 100\%)$, plus one overload segment $[100\%, \infty)$. Given the two ECN bits, we can partition the whole load factor range into the four segments as shown in Figure 3. (To get the complete benefit of two bits, hereafter we fully utilize the two ECN bits to encode four, instead of three, load regions, as described earlier). We choose the following four ranges for quantizing and encoding the load factor ρ_l: [2]

- Low-load region: $\hat{\rho}_l = 50\%$ when $\rho_l \in [0\%, 50\%)$;
- Medium-load region: $\hat{\rho}_l = 80\%$ when $\rho_l \in [50\%, 80\%)$;
- High-load region: $\hat{\rho}_l = 100\%$ when $\rho_l \in [80\%, 100\%)$;
- Overload region: $\hat{\rho}_l > 100\%$ when $\rho_l \in [100\%, \infty)$.

The quantized load factors $\hat{\rho}_l$ can be represented by the same number of two-bit codes $\hat{\rho}_l^c$, i.e., $\hat{\rho}_l^c = (00)_2, (01)_2, (10)_2$ and $(11)_2$ for $\hat{\rho}_l = 50\%, 80\%, 100\%$ and $\hat{\rho}_l > 100\%$, respectively. The encoded two-bit load factor $\hat{\rho}_l^c$ is carried in the data packet IP header from the source to the destination and then echoed back to the source by the ACK packets. In a network with multiple links, for each flow i, we need to convey to the source the maximal quantized load factor $\max_{l \in L_i}(\hat{\rho}_l)$ of all those links on the flow i's forward path $L_i = \{l \mid i$ traverses link $l\}$. Thus, each router updates the load factor in a data packet header if its output link's load factor is higher than the load factor already encoded in the packet header. As the quantized load factor $\hat{\rho}_l$ is a non-decreasing function of the raw load factor ρ_l, and this order is also preserved by the code $\hat{\rho}_l^c$, i.e., $\hat{\rho}_1^c < \hat{\rho}_2^c \Leftrightarrow \hat{\rho}_1 < \hat{\rho}_2 \Rightarrow \rho_1 < \rho_2$, the comparison

[2]These three split points work very well in our ns2 simulations that cover a wide range of network parameter settings in capacity, delay, and number of flows, etc. They are chosen based on the following considerations: 1) 100% is an obvious separator between the underload regions and the overload region; 2) To give a safety margin for the system to operate in the AIMD mode in its steady state, 80% is used to separate the medium-load region and the high-load region since it is slightly less than the MD parameter $\beta = 0.875$ defined in Section III-B; 3) We choose 50% to separate the low-load region and the medium-load region because $\xi_1(80\%) = \kappa_\rho \cdot \xi_1(50\%)$ where $\kappa_\rho = 0.25$ is used for $\hat{\rho}_l = 80\%$ in Equation (29) in Section III-E.2 and $\xi_1(\hat{\rho}_l)$ is defined by Equation (10) in Section III-B. More discussion on this is in Section III-B.

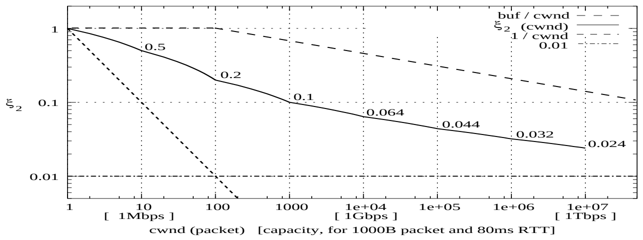


Fig. 4. The MI parameter ξ_2 is a pairwise linear function of $\log(cwnd)$. The line $buf/cwnd$ roughly represents the declining trend of router buffer size compared with $cwnd$ (or BDP). AI corresponds to $1/cwnd$. STCP uses a fixed MI parameter 0.01 [30]. Note the logarithmic scale in both axes. Also note x-axis roughly spans from 100Kbps to 1Tbps.

operation on the load factor ρ_l at any link l can directly operate on the two-bit code $\hat{\rho}_l^c$.

B. End Host Window Increase/Decrease Algorithms

The end host switches its window-based control between MI/AI/MD depending on the load factor feedback. To simplify the discussion, we consider a single-link network shared by two flows f_1 and f_2, whose RTTs are equal to the measurement period of the link load factor, i.e., $rtt_1 = rtt_2 = t_\rho$. So the flows have synchronous feedback and their control intervals are also in sync with the link load factor measurement. We will handle the case of non-synchronous feedback and control due to heterogeneous RTT in Section III-C.

Following the simplified VCP model described in Section II-C, if the end host receives an encoded load factor $\hat{\rho}_l^c$ that represents the low-load or medium-load regions, it increases its congestion window $cwnd$ using MI — this guarantees to exponentially probe any amount of available bandwidth. For high-load regions, the end host increases $cwnd$ using AI — together with MD, this guarantees the convergence toward fairness. Otherwise, upon getting the first overload signal, the end host immediately decreases its $cwnd$ using MD, followed by AI for the remaining time in the same RTT. So, at any time t, we apply one of the following algorithms:

$$\begin{aligned} \texttt{MI}: \quad cwnd(t+rtt) &= cwnd(t) \times (1+\xi) && (7)\\ \texttt{AI}: \quad cwnd(t+rtt) &= cwnd(t) + \alpha && (8)\\ \texttt{MD}: \quad cwnd(t+\delta t) &= cwnd(t) \times \beta && (9) \end{aligned}$$

where $rtt = t_\rho$, $\delta t \to 0+$, $\xi > 0$, $\alpha > 0$ and $1 > \beta > 0$.

Now, we describe how to set the parameters ξ, α and β. First, as discussed in the Section III-A, we require an MD parameter β that does not force the two flows to make a quantum jump from high utilization to the medium-load or low-load regions. Once the system reaches high utilization, the goal is to retain the system in this regime (if no capacity or demand change) and achieve fairness amongst the two flows by using AIMD. Hence we choose $\beta = 0.875$ (instead of 0.5 used by TCP), a value larger than 80% which separates the high-load and medium-load regions (see Section III-A). Second, we set the AI parameter $\alpha = 1.0$. For high BDP networks, this makes AI probing extremely slow. However, it does not matter much since AI means the network has already entered the high-utilization region where efficiency does no longer suffer from the slow probing speed. Third, to set the MI parameter ξ, we need to handle two factors: the intra-flow factor and the inter-flow factor.

Intra-flow factor: With a current load factor $\hat{\rho}_l$, in the next RTT each flow needs to increase its $cwnd$ to fill at least part of the spare capacity that is proportional to $1 - \hat{\rho}_l$. And because the $cwnd$ change uses MI, and it is based on the current sending rate which is proportional to $\hat{\rho}_l$, we get:

$$\xi_1(\hat{\rho}_l) = \kappa \cdot \frac{1-\hat{\rho}_l}{\hat{\rho}_l} \tag{10}$$

where κ controls the speed to converge toward full utilization. We set $\kappa = 0.25$ and thus $\xi_1(50\%) = 0.25$ and $\xi_1(80\%) = 0.0625$.

Inter-flow factor: For multiple flows, we argue that it is better to set ξ as a decreasing function than a constant value, and its decrease should be slower than $O(\frac{1}{cwnd})$. To see why is this consider two extreme cases. If we use a constant value $\xi = c$, then MI is increasingly unfair [3] to low-rate flows, and becomes increasingly bursty causing larger queues and potentially packet losses. On the other hand, if we chose $\xi = \frac{1}{cwnd}$, then MI degenerates to AI. As a result, we choose a function that is piecewise linear in $\log(cwnd)$: [4]

$$\xi_2(cwnd) = a - b\log(cwnd) \tag{11}$$

and $\xi_2(cwnd) > 0$. Instead of setting a and b, we choose a set of specific values and interpolate the function $\xi_2(cwnd)$ shown in Figure 4. For example, one set of values we use are: [5]$\xi_2(1) = 1.0$, $\xi_2(10) = 0.5$, $\xi_2(10^2) = 0.2$, $\xi_2(10^3) = 0.1$, $\xi_2(10^4) = 0.064$, $\xi_2(10^5) = 0.044$, $\xi_2(10^6) = 0.032$ and $\xi_2(10^7) = 0.024$. This setting has also taken into account the relatively decreasing trend of the router buffer size compared with BDP. For efficient implementation, $\xi_2(cwnd)$ can be computed off-line and organized as a lookup table indexed by $\lceil cwnd \rceil$, where $\lceil x \rceil$ is the smallest integer that is not less than x. Finally, we combine the above two factors together:

$$\xi = \min(\,\xi_1(\hat{\rho}_l),\ \xi_2(cwnd)). \tag{12}$$

C. Handling RTT Heterogeneity by Parameter Scaling

So far we have considered the condition of $rtt_1 = rtt_2 = t_\rho$. Now we relax this condition on the previous model. The flows have heterogeneous RTTs that usually differ from the link load factor measurement interval as well. RTT substantially affects the performance of window-based congestion control. To offset the impact of RTT heterogeneity, we need to generalize the end host algorithms in Equations (7)$\sim$(9).

[3]Although it is not required to handle fairness in the MI phase, applying the coming Equation (11) improves the fairness convergence speed of VCP. This is a necessary compensation for the slow fairness convergence of the end-host based AIMD which VCP applies. Refer to discussion in Section III-D.

[4]The logarithmic function $\xi_2(cwnd)$ is a middle-ground choice because $(-\log(cwnd))' = \frac{-1}{cwnd}$, while $(const)' = 0$ and $(\frac{1}{cwnd})' = \frac{-1}{cwnd^2}$.

[5]To illustrate how we come up with these values, consider the following numerical examples: i) Given this setting, it takes less than 70 RTT to grow $cwnd$ from 1 to 10^4 packets, and less than 210 RTT from 10^4 to 10^7; ii) If two flows $f1$ and $f2$ start with different initial congestion windows $cwnd_{f1}(0) = 10 \cdot cwnd_{f2}(0)$, but $\xi_{f2} - \xi_{f1} = 0.033$, then after about $t = 70 \cdot rtt$ we will reach $cwnd_{f1}(t) = cwnd_{f2}(t)$.

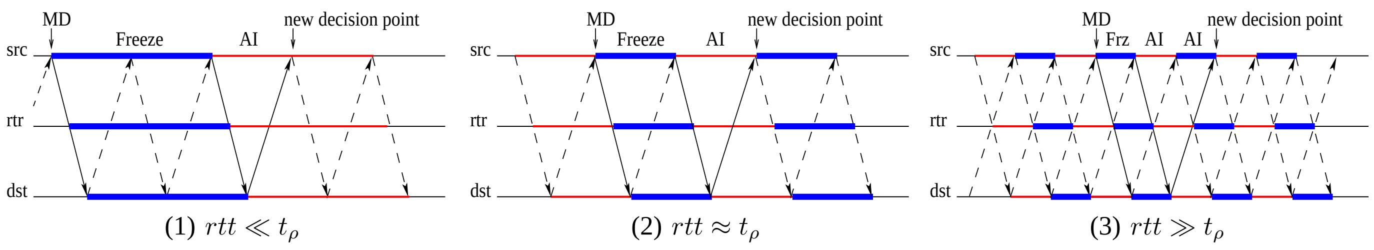


Fig. 5. After each MD that is triggered by the first overload signal, the source host freezes its $cwnd$ for t_ρ (for the bottleneck to generate a new load factor) and then applies AI for one rtt (for the new load factor to come back), regardless of the remaining encoded load factors during these two periods.

We first consider MI and AI which last for an entire RTT. If we scale the parameters of the MI/AI algorithms in Equations (7)-(8) for a flow with its RTT value and a common base t_ρ: [6]

$$\texttt{For MI}: \quad \xi \quad \leftarrow \quad (1+\xi)^{\frac{rtt}{t_\rho}} - 1\,, \tag{13}$$

$$\texttt{For AI}: \quad \alpha \quad \leftarrow \quad \alpha \cdot \frac{rtt}{t_\rho}\,, \tag{14}$$

then during any time period S, with Equation (7)/(8), the flow will MI/AI an amount that is independent of the length of its RTT, given synchronous feedback:

$$\texttt{MI}: \quad cwnd(t+S) \;=\; cwnd(t) \times (1+\xi)^{\frac{S}{t_\rho}}\,, \tag{15}$$

$$\texttt{AI}: \quad cwnd(t+S) \;=\; cwnd(t) + \alpha \cdot \frac{S}{t_\rho}\,. \tag{16}$$

This scaling compensates RTT heterogeneity among flows, which would otherwise cause small-RTT flows to gain a significant advantage than large-RTT flows, either in MI or in AI. In the other word, the value of RTT is decoupled from deciding the congestion window increase speed.

For MD which is an impulse-like operation, since Equation (9) is not affected by the length of RTT, it needs no change. However, the behavior after the MD needs to accommodate the length of t_ρ. As illustrated in Figure 5, upon getting the first load factor feedback that signals congestion (i.e., $\hat{\rho}_p^c = (11)_2$), the end host immediately cuts its congestion window $cwnd$ using MD. Next, it freezes its $cwnd$ firstly for a time period of t_ρ (for all the three cases shown in the figure) such that the congested router can generate a new load factor based on the source sending rate right after the MD. Then the end host applies AI for one RTT, the time that it needs to get the new load factor feedback from the destination ACK packet. Based on this new load factor, a new decision can be made. Of course, during these two time periods the router could be congested again (e.g., because of incoming traffic). That is where the router buffer space comes into play. As the last resort, if the route buffer overflows, packet gets lost and another loss-induced MD will be performed, preventing congestion collapse. Although Figure 5 illustrates only either $t_\rho = n \cdot rtt$ or $rtt = n \cdot t_\rho$ for an integer n, this analysis can be generalized to the non-integer cases.

However, RTT is the one which determines how long it takes to get the load factor back to the source host. Consider two flows f and F with RTT values such that $rtt_f \ll rtt_F$, which share the same bottleneck l. This means that, after a load factor $\hat{\rho}_l^c$ is generated, flow F will receive it much later than flow f. We focus on the situation where the load factor triggers a mode switch that might have a negative impact on fairness, i.e., when $\hat{\rho}_l^c = (10)_2$. Flow f has switched from MI to AI, but flow F still keeps doing MI for a time period of up to rtt_F long, since it has not received that load factor. During such a time interval, with the RTT scaling according to Equation (15), flow F will increase its congestion window from $cwnd$ to $cwnd \cdot (1+\xi)$ and gain an unfair share. To limit this effect, we bound $\frac{rtt}{t_\rho}$ above by σ_{mi}. Of course, this upper bound will cause the large-RTT flows to probe at a slower speed in MI than small-RTT flows. Fortunately, according to the design guideline in Section II-B, it is not required to achieve fairness in the MI phase. We will revisit this issue in Section V-A.

[6] Equation (13) is the solution for $1+\xi = (1+\xi_{scaled})^{\frac{t_\rho}{rtt}}$ where the right-hand side is the MI amount during a time interval t_ρ of a flow with RTT value rtt. Similarly, Equation (14) is obtained by solving $1+\alpha = 1+\frac{t_\rho}{rtt} \cdot \alpha_{scaled}$.

D. Weighted Bandwidth Sharing via Window-based Control

Even if the window-based control algorithm achieves fair $cwnd$ allocation, this does not directly translate to fair bandwidth allocation. The standard AIMD algorithm has to be extended to provide fair (or even differentiated) bandwidth sharing. We first consider the AIMD rate control. Consider two flows that share the same bottleneck(s) and receive synchronous congestion signals. Flow i starts from an initial sending rate $r_i(0)$, where $i = 1$ or 2. If each of them have an AI parameter α_{ri} and an identical MD parameter β, then at the end of the first congestion epoch that includes m rounds of AI and one round of MD: [7]

$$r_i(1) = \beta \cdot (r_i(0) + m \cdot \alpha_{ri}). \tag{17}$$

The rate ratio between the two flows is:

$$\Lambda_{1,2}^r(1) = \frac{r_1(1)}{r_2(1)} = \frac{r_1(0)/m + \alpha_{r1}}{r_2(0)/m + \alpha_{r2}}, \tag{18}$$

Hence, when the number of the congestion epochs M is large enough, the two flows will have a weighted share that is proportional to their rate AI parameters α_{ri}:

$$\Lambda_{1,2}^r(M) = \frac{r_1(M)}{r_2(M)} \;\rightarrow\; \frac{\alpha_{r1}}{\alpha_{r2}}. \tag{19}$$

[7] We implicitly assume $m \gg 1$. As discussed in Section III-B, since the AI parameter α is refrained from being set as a large number to probe bandwidth quickly – which is now the job of the MI phase – we can always find an α for any given MD parameter β such that $m = \frac{1-\beta}{\alpha} \cdot cwnd \gg 1$. For example, given $\beta = 0.875$, set $\alpha = 1.0$ makes the assumption true for high BDP networks (where, e.g., $cwnd \gg 30$). An even smaller number can be used for α to make this assumption virtually always valid in practice.

The weighted rate difference at the end of the M-th congestion epoch is:

$$\Delta^r_{1,2}(M+1) = \beta \cdot \Delta^r_{1,2}(M) \tag{20}$$

where

$$\Delta^r_{1,2}(M) = \frac{r_1(M)}{\alpha_{r1}} - \frac{r_2(M)}{\alpha_{r2}} \tag{21}$$

for $M = 0, 1, 2, \ldots$. Obviously here β decides the speed to converge onto the target weighted share. For instance, given $\beta = 0.875$, it takes about five congestion epochs to eliminate half of the weighted rate difference.

Fairness gets improved per congestion *epoch* with the end-host based AIMD algorithm. While a network-based AIMD scheme like XCP improves fairness in every *round* of control. Consequently, an end-host based scheme like VCP will be significantly slower than XCP in the fairness convergence speed. See Section IV-D for simulation results.

The above algorithm is based on the AIMD rate control. To achieve the same goal, it has to be tailored for any window-based scheme. It is also orthogonal to the handling of heterogeneous RTTs in Section III-C. For a complete solution we need to integrate these two factors. For example, if we want to achieve a weighted bandwidth share r for a flow with a weight w, i.e., $r \propto w$, considering $cwnd = r \cdot rtt$ and Equations (14) and (19), we end up with scaling the window AI parameter α for a typical RTT $t_d = 100$ms, and $t_\rho = 200\text{ms} = 2\,t_d$ with:

$$\texttt{For AI}: \alpha \;\leftarrow\; \alpha \cdot \frac{rtt^2}{t_\rho t_d} \cdot w \;=\; \frac{\alpha}{2} \cdot (\frac{rtt}{t_d})^2 \cdot w\,. \tag{22}$$

This scaling, however, might introduce a significant amount of bursty traffic within one RTT if $rtt \gg t_d$ or w is huge and therefore cause packet loss. To limit this effect, we impose an upper bound σ_{ai} on $w \cdot (\frac{rtt}{t_d})^2$ such that the burstiness can be effectively absorbed by the router buffer. Of course, this upper bound will cause the large-RTT flows to receive a lower bandwidth share than otherwise, as shown in Section IV-E.

E. The VCP Protocol

Now, we are in the position to describe in detail the operations performed by the VCP routers and the VCP end hosts as well as how they interact.

1) The Router: The VCP router computes and encodes a load factor based on the number of incoming packets and the average queue for each output link. Then, it updates the encoded load factor in the IP header if the value computed by the router is larger than the one carried by the packet.

VCP Router Algorithm

R.1) For each incoming packet of size s_p, update a counter:

$$\lambda_l \;\leftarrow\; \lambda_l + s_p \qquad //\; Count \tag{23}$$

R.2) When the queue sampling timer t_q fires at time t:

$$\widetilde{q}_l \;\leftarrow\; \texttt{EWMA}(\widetilde{q}_l,\, q(t)) \qquad //\; Average \tag{24}$$

i.e., update the low-pass filtered queue measurement $\widetilde{q}_l$ with an Exponentially Weighted Moving Average (EWMA) similar to what is used in RED. We set $t_q = 10$ms;

R.3) When the load factor measurement timer t_ρ fires:

$$\begin{aligned}
\rho_l &= \frac{\lambda_l + \kappa_q \cdot \widetilde{q}_l}{\kappa_l \cdot c_l \cdot t_\rho} && //\; Measure && (25)\\
\hat{\rho}^c_l &\leftarrow \texttt{encode}(\rho_l) && //\; Encode && (26)\\
\lambda_l &\leftarrow 0 && //\; Reset && (27)
\end{aligned}$$

where $\kappa_q = 0.5$, $\kappa_l = 0.98$, $t_\rho = 200$ms, c_l is the link capacity, and the encoding function is defined in Section III-A;

R.4) For each dequeued data packet p that carries an encoded load factor $\hat{\rho}^c_p$ from upstream:

$$\hat{\rho}^c_p \;\leftarrow\; \max(\hat{\rho}^c_l,\, \hat{\rho}^c_p) \qquad //\; Tag \tag{28}$$

2) The End Hosts: The VCP receiver is the same as the TCP Reno receiver, except that it copies the encoded two-bit load factor $\hat{\rho}^c_p$ from the data packet to its corresponding ACK packet. The VCP sender builds upon the TCP Reno sender. It behaves like TCP Reno when packet loss happens. The VCP sender also initializes the encoded load factor $\hat{\rho}^c_p$ in the data packet IP header to $(00)_2$. It switches its window-based control between MI/AI/MD according to the encoded load factor $\hat{\rho}^c_p$ received in the ACK packet. This switching is performed as follows.

VCP Sender Algorithm

S.1) If the load factor is low/moderate ($\hat{\rho}^c_p = (0X)_2$), do:

$$\begin{aligned}
\xi &= \min(\xi_1(\hat{\rho}_l),\; \kappa_\rho \cdot \xi_2(cwnd)) && \\
inc &= (1.0 + \xi)^{\min(\frac{srtt}{t_\rho},\; \sigma_{mi})} - 1.0 && //\; Scaling \\
cwnd &\leftarrow cwnd + inc && //\; MI \quad (29)
\end{aligned}$$

where $\hat{\rho}_l = 50\%$, $\kappa_\rho = 1.0$ for $\hat{\rho}^c_p = (00)_2$ and $\hat{\rho}_l = 80\%$, $\kappa_\rho = 0.25$ for $\hat{\rho}^c_p = (01)_2$, ξ_1 and ξ_2 are defined in Equations (10) and (11) in Section III-B, respectively, $srtt$ is the smoothed RTT measurement in ms, $t_\rho = 200$ms, and the MI burstiness limiter $\sigma_{mi} = 2.5$;

S.2) If the load factor is high ($\hat{\rho}^c_p = (10)_2$), do:

$$\begin{aligned}
inc &= \alpha \cdot \min((\frac{srtt}{t_d})^2 \cdot w, \sigma_{ai}) && //\; Scaling \\
cwnd &\leftarrow cwnd + inc\,/\,cwnd && //\; AI \quad (30)
\end{aligned}$$

where $\alpha = 1.0$, $t_d = 100$ms, the AI burstiness limiter $\sigma_{ai} = 10.0$, and the weight w is settable with a default value 1.0;

S.3) For the first "overload" load factor ($\hat{\rho}^c_p = (11)_2$), cut the congestion window once immediately with:

$$cwnd \;\leftarrow\; \max(1.0,\, \beta \cdot cwnd) \qquad //\; VCP\text{–}MD \tag{31}$$

where $\beta = 0.875$. Then, for a time period of length $t_\rho + srtt$, first freeze $cwnd$ for one t_ρ and second follow S.2 for one $srtt$, regardless of the value of the remaining load factors

TABLE I
VCP PARAMETER SETTINGS

Parameter	Value	Meaning
t_ρ	200 ms	the link load factor measurement interval
t_q	10 ms	the router queue sampling interval
κ_l	0.98	the link target utilization
κ_q	0.5	how fast to drain the router steady queue
t_d	100 ms	the typical RTT value of a flow
κ	0.25	how fast to probe the available bandwidth
κ_ρ	1.0 / 0.25	the MI adjuster, for $\hat{\rho}_p^c = (00)_2\,/\,(01)_2$
α	1.0	the AI parameter
β	0.875	the MD parameter
σ_{mi}	2.5	the traffic burstiness limiter for MI
σ_{ai}	10.0	the traffic burstiness limiter for AI
w	1.0	the default weight value (may change)

during this time period. (Even we do only one VCP-MD in this time period, since VCP builds upon TCP Reno, the loss-triggered MD might be performed as well. This happens when the VCP-MD rate cut is not sufficient to bring the network out of congestion.)

IV. PERFORMANCE EVALUATION

In this section, we use extensive ns2 simulations to evaluate the performance of VCP under different network topologies, configurations and traffic load. For the purpose of comparison, we use similar network configurations as used in the XCP paper [27]. Our simulations cover a wide range of network scenarios, such as link capacities in [1.5Mbps, 4Gbps], round trip propagation delays in [10ms, 1.4s], numbers of long-lived, FTP-like flows in [1, 1000], and arrival rates of short-lived, web-like flows in [$1s^{-1}$, $1000s^{-1}$]. In all the simulations we use two-way traffic with congestion resulted in the reverse path. The bottleneck buffer size is always set to one bandwidth-delay product. The data packet size is 1000 bytes, while the ACK packet is 40 bytes. All the simulations run for at least 120s to ensure that the system has reached steady state. The average utilization statistics neglect the first 20% of simulation time. For all the time-series graphs, utilization and throughput are averaged over 500ms interval, while queue length and congestion window are sampled every 20ms. We use a *fixed* set of VCP algorithm parameters shown in Table I for all the simulations in this paper.

Our results demonstrate that, for a wide range of network scenarios, particularly high BDP scenarios, VCP is able to achieve comparable performance as XCP, i.e., exponential convergence onto high utilization, low persistent queue length, negligible packet drop rate and reasonable fairness, except that its fairness convergence is significantly slower than XCP. Due to utter complexity of the Internet and limitations of the simulation approach [16], although we have painstakingly run more than 160 simulations for this paper and tried to make them representative (e.g., two-way traffic, different RTT, rough timer, parameter settings, etc.), we do not claim that these simulations cover all, or even a major part of, the real-world scenarios. Conclusions are made to the extent that is covered by them. Nevertheless, we believe the results below still give us substantial confidence on the performance of VCP.

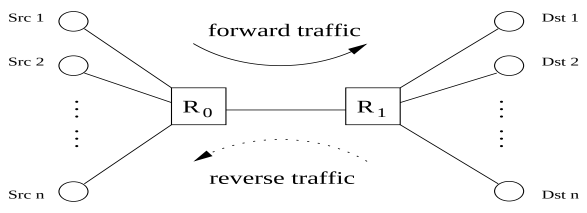


Fig. 6. A single-bottleneck topology with two-way traffic.

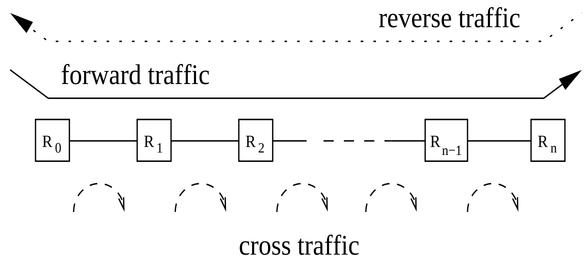


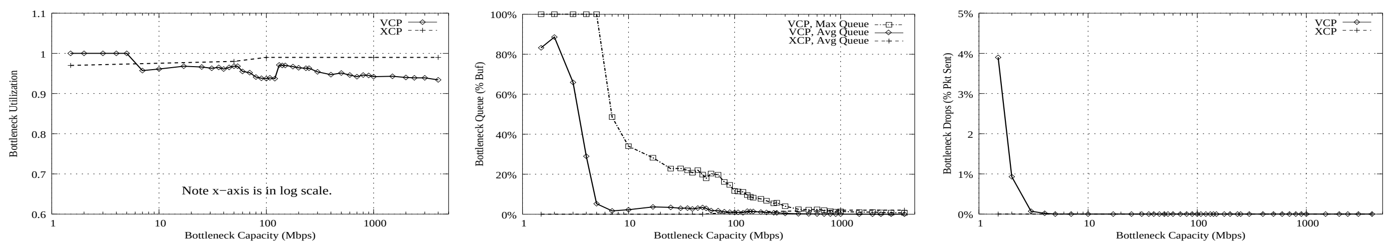
Fig. 7. A multiple-bottleneck parking-lot topology ($n = 9$ is used).

A. One Bottleneck

Firstly we evaluate the performance of VCP for a single bottleneck shown in Figure 6, where the capacity, the round-trip propagation delay, the number of FTP flows and the arrival rate of web flows may change. The basic setting is an 150Mbps link with 80ms RTT where 50 FTP flows are on the forward path. There are 50 FTP flows on the reverse path. The changes are then made over this basic setting to evaluate the impact of each network configuration parameter. All the simulations in this subsection run for 120s or 300 RTTs, whichever is larger. For comparison purpose, in this subsection we use exactly the same network configurations as in Section 5.2 of [27], from which we reuse the XCP results in Figures 8∼11. (Note XCP does not provide maximal queue statistics.)

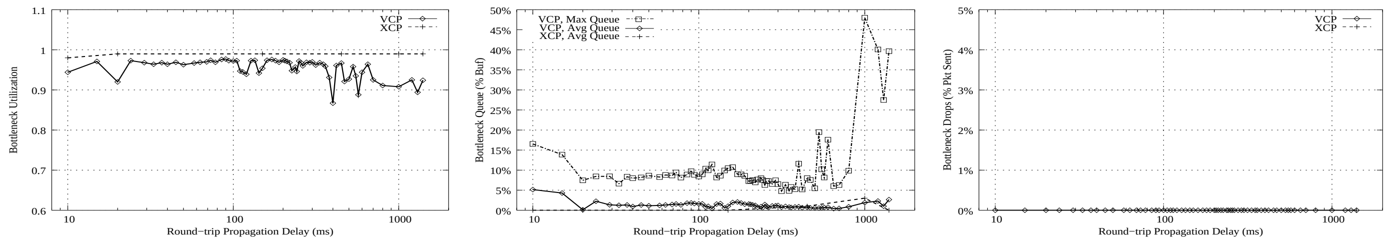
Impact of Bottleneck Capacity: In this simulation, we vary the bottleneck capacity from 1.5Mbps to 4Gbps, but fix other parameters. Figure 8 demonstrates that for all the cases, VCP always keeps high utilization. As capacity increases, for the cases where the bottleneck capacity is larger than 10Mbps (i.e., per-flow $cwnd$ of 2 packets), VCP's bottleneck maximal queue and average queue both decrease towards about 1% of the bottleneck buffer size and no packet gets dropped. (All the queue statistics in this subsection is in percentage of bottleneck buffer size.) When the capacity is extremely low, e.g., 1.5Mbps (i.e., per-flow $cwnd$ of 0.3 packet), the bottleneck average queue significantly increases to 83% of the buffer size, resulting in 4% packet loss. This is due to the AI parameter $\alpha = 1.0$ which is too large for such low capacity. Comparing VCP to XCP, we find the utilization gap is at most 6%, indicating that overall VCP achieves similar performance as XCP.

Impact of Feedback Delay: We fix the bottleneck capacity at 150Mbps and vary the round-trip propagation delay from 10ms to 1400ms. As shown in Figure 9, when we vary the delay between 10ms and 400ms, we notice that the bottleneck average utilization is mostly around 92%∼97%, the average (maximal) queue is always less than 3% (12%) of the buffer



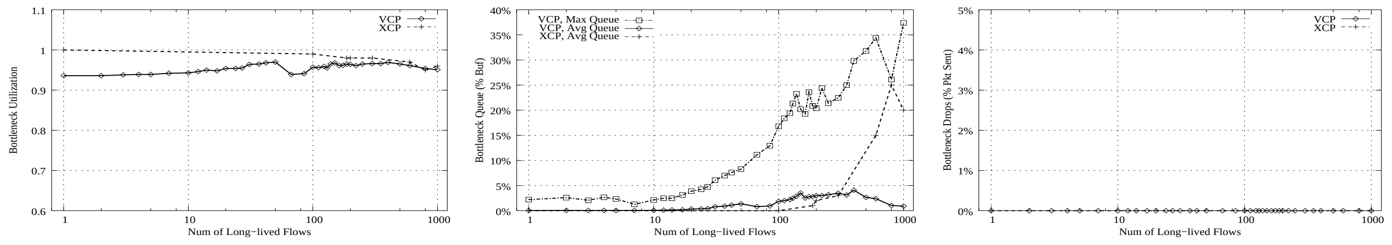
(1) Bottleneck Average Utilization (2) Bottleneck Maximal/Average Queue (3) Bottleneck Drops

Fig. 8. VCP with the bottleneck capacity ranging from 1.5Mbps to 4Gbps. (1) The bottleneck average utilization is always higher than 93%; (2-3) For most cases ($\geq$ 10Mbps), the bottleneck average (maximal) queue is lower than 4% (34%) of the buffer size and there is no packet drop. When the capacity is lower than 5Mbps (i.e., the per-flow congestion window is less than one packet), the bottleneck average queue is large and a small percentage of packets (e.g., 4% for the case of 1.5Mbps) get lost. Overall, VCP performs slightly worse than XCP. Note the logarithmic scale of the x-axis in these figures.



(1) Bottleneck Average Utilization (2) Bottleneck Maximal/Average Queue (3) Bottleneck Drops

Fig. 9. VCP with the round-trip propagation delay ranging from 10ms to 1400ms. (1) The bottleneck average utilization is mostly higher than 91%, except three cases: 86.7% for 400ms, 88.8% for 570ms, and 89.4% for 1300ms; (2) The bottleneck average queue is always lower than 5% (mostly around 1%$\sim$2%) of the buffer size. The maximal queue gets higher for delays larger than 1000ms, but is still less than 50% buffer size (see the text for explanation); (3) There is no packet loss. The bottleneck average utilization performance gap between VCP and XCP could be as large as 12%, but the overall performance of VCP is still comparable with XCP.



(1) Bottleneck Average Utilization (2) Bottleneck Maximal/Average Queue (3) Bottleneck Drops

Fig. 10. VCP with the number of long-lived, FTP-like flows ranging from 1 to 1000. (1) The bottleneck average utilization is always higher than 93%; (2) The bottleneck average (maximal) queue is always less than 5% (38%) of the buffer size; (3) There is no packet drop. VCP performs slightly worse than XCP in the bottleneck utilization. When the bottleneck is heavily multiplexed, e.g, with 800+ flows, VCP even slightly outperforms XCP.

size. When the delay exceeds 400ms, VCP performs worse than the lower delay cases, with lower utilization and much higher maximal queue length. (Note the logarithmic scale of the x-axis in the figure.) The reason is that when the load factor measurement interval $t_\rho = 200$ms is much lower than the RTTs, the link load condition measured each t_ρ is not reliable due to the busty nature of window-based control. This can be counteracted by increasing t_ρ; but the tradeoff is the link load measurement will be less responsive. Note in this simulation all the flows have the same RTT, and we believe the case for all the flows having similar large RTTs (> 400ms) is not a common condition in practice. Nevertheless, even for the large RTT cases, the worst utilization is 86%, the highest maximal queue is still less than 50% of the bottleneck buffer size, the persistent queue length is around 2% buffer size and there is no packet drops. Comparing to XCP, VCP's utilization could be 12% less than XCP. But their average queue length is similar and both of them drop no packet.

Impact of Number of Long-lived Flows: With an increase in the number of forward FTP flows (Figure 10 shows the results), we notice that the whole traffic gets more bursty, as shown by the increasing trend of the bottleneck maximal queue. However, even when the network is very heavily multiplexed by 1000 flows (i.e., the average per-flow $cwnd$ equals to only 1.5 packets), the maximal queue is still less than 38% of the buffer size. For all the cases, VCP behaves very comparable to XCP.

Impact of Short-lived Traffic: Finally we add web traffic into the network. These short-lived web flows arrive according to the Poisson process, with the average arrival rate varying from 1/ s to 1000/ s. Their transfer size obeys the Pareto distribution with an average of 30 packets (corresponding to the parameter $shape\_ = 1.35$ in ns2). This setting is consistent with the real-

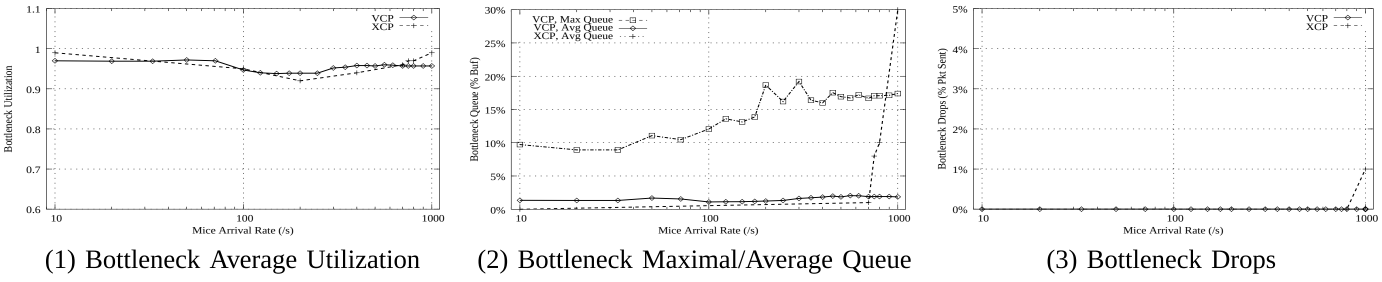


Fig. 11. Similar to XCP, VCP remains efficient with the short-lived, web-like flows arriving/departing at a rate from 1/s to 1000/s. (1) The bottleneck average utilization is always higher than 93%; (2) The average (maximal) queue is always less than 3% (20%) of the buffer size; (3) There is no packet drop.

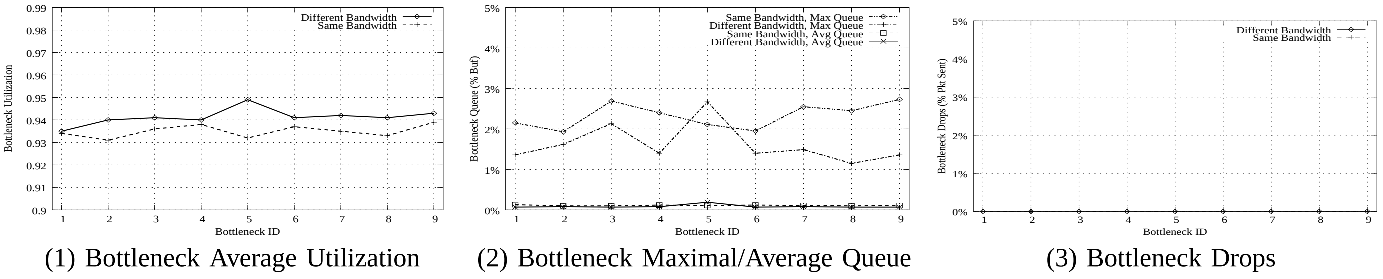


Fig. 12. VCP with multiple congested bottlenecks. When either all the links have the same capacity (100Mbps), or the middle link #5 has lower capacity (50Mbps) than the rest of the links, VCP consistently achieves high utilization, low persistent queue and zero packet drop on all the bottlenecks.

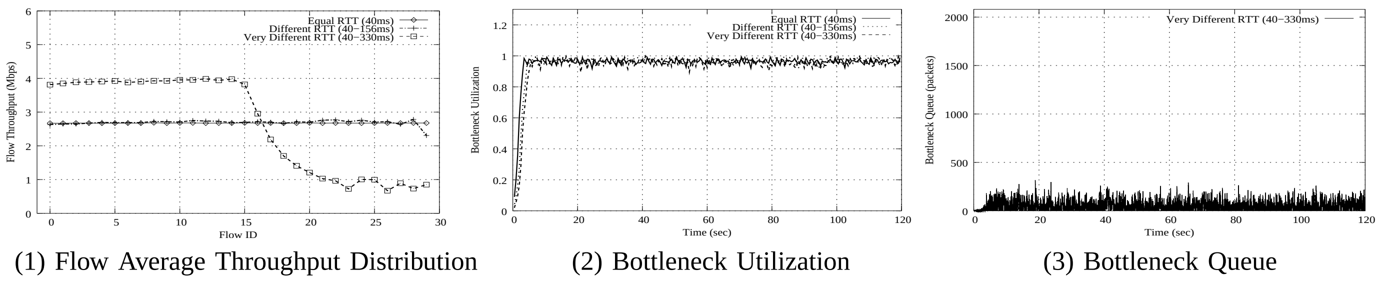


Fig. 13. To some extent, VCP distributes bandwidth fairly among competing flows with either equal or different RTTs. (1) The bottleneck bandwidth gets evenly allocated to all the flows if the RTT ratio is within 4. Beyond that, flow throughput distribution spreads wider when the RTT heterogeneity is significant, with the throughput ratio of up to 5 given the RTT ratio of up to 8; (2) The bottleneck utilization keeps higher than 95% after the starting period; (3) Even for the case of significant RTT heterogeneity (i.e., 40ms$\sim$330ms), the bottleneck queue remains very low, given the buffer size of 2080 packets. Compared with the results in [27], VCP performs much better than TCP+RED/REM/AVQ/CSFQ and is close to, though not as good as (for the 40ms$\sim$330ms case), XCP.

world web traffic model [10]. As shown by Figure 11, the bottleneck always maintains high average utilization with low queue length and zero packet drop, quite similar to XCP.

Across a wide range of network configuration, these simulations demonstrate that VCP can achieve comparable performance to XCP in a one-bottleneck scenario. Note this result is achieved with a fixed set of VCP parameters, indicating the robustness of the VCP algorithm.

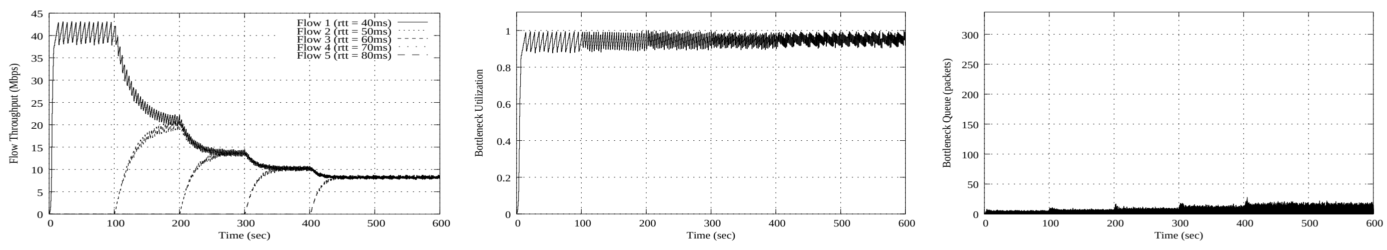
B. Multiple Bottlenecks

Next, we study the performance of VCP using a more complex network topology with multiple bottlenecks. For this purpose, we use a classic parking-lot topology with nine links, as shown in Figure 7. All the links have a 20ms one-way propagation delay. There are 50 FTP flows traversing all the links in the forward direction, and 50 FTP flows traversing all the links in the reverse direction as well. In addition, each individual link has 5 cross FTP flows traversing in the forward direction. We run two simulations. First, all the links have 100Mbps capacity. Second, the middle link #5 has the smallest capacity of only 50Mbps, while all the others have the same capacity of 100Mbps.

Figure 12 shows that for both cases, VCP performs as good as in the single-bottleneck scenarios. When all the links have the same capacity, VCP achieves higher than 93% average utilization, less than 0.2%-buffer-size average queue length and zero packet drops at all the bottlenecks. When we lower the capacity of one of the links in the topology, the average utilization increases to 95%, with the largest maximal queue representing only 2.7% of the buffer size.

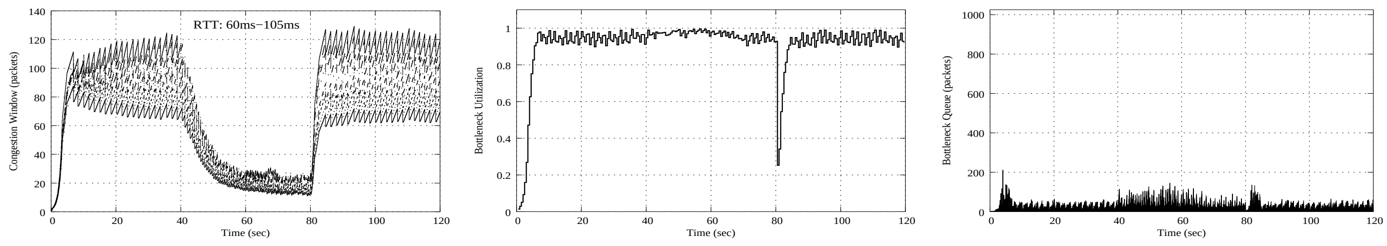
C. Fairness

TCP flows with different RTTs achieve bandwidth allocation that is proportional to $1/rtt^z$ where $1 \le z \le 2$ [36]. VCP alleviates this issue to some extent. Here we look at the RTT-fairness of VCP. We have 30 FTP flows sharing a single



(1) Flow Throughput (2) Bottleneck Utilization (3) Bottleneck Queue

Fig. 14. VCP converges onto good fairness, high utilization and small queue. However, its fairness convergence takes significantly longer time than XCP. (1) Every 100s, a new flow comes in and the network converges onto a new fair bandwidth allocation; (2) The bottleneck utilization keeps higher than 93% after the starting period; (3) The bottleneck queue remains low throughout the simulation, comparing to its buffer size of 337 packets.



(1) Flow Congestion Window (2) Bottleneck Utilization (3) Bottleneck Queue

Fig. 15. VCP is robust against and responsive to sudden, considerable traffic demand change. Here 40 FTP flows join the system of 10 FTP flows (with heterogeneous RTTs of 60ms$\sim$105ms) at 40s and leave at 80s. (1) The 10 flows' congestion windows adapt quite responsively to the sudden demand changes at 40s and 80s; (2) The bottleneck quickly ramps up to high utilization in 4 seconds after the departure of the 40 flows at 80s; (3) The bottleneck queue gets only slightly more bursty with all the 50 flows active. It remains small throughout the simulation, comparing to the buffer size of 1025 packets.

90Mbps bottleneck, with 30 FTP flows on the reverse path. We perform three sets of simulations for three cases: (a) same RTT; (b) small RTT difference; (c) huge RTT difference. We will see that VCP is able to allocate bottleneck bandwidth fairly among competing flows, as long as their RTTs are not significantly different. This capability degrades as the RTT heterogeneity increases.

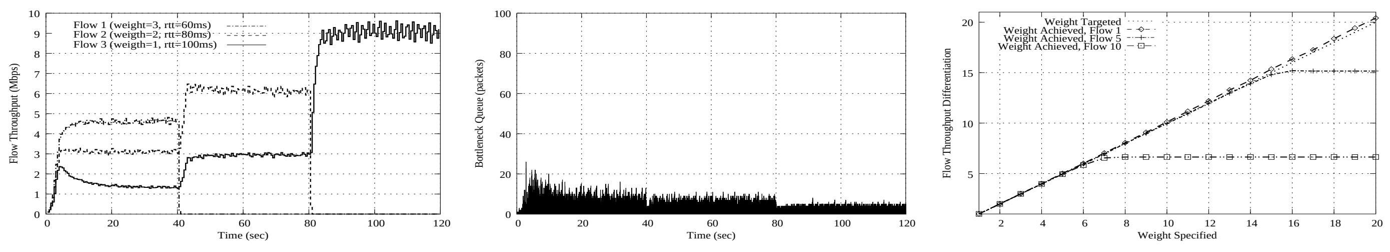
First, all the forward FTP flows have a common RTT of 40ms. As shown in Figure 13, the link capacity is very evenly distributed among all the flows. An average utilization of 96% is achieved with an average queue of 5 packets (less than 1.2% buffer size), maximal queue 39 packets (or 8.7% buffer size) and no packet drops. Second, when the flows have small RTT difference, we notice that the bandwidth sharing is again very fair. And other performance matrices are also similarly good. Third, the flows have significantly different RTTs - 40ms to 330ms (i.e., RTT ratio of about 8). This seems to stress the protocol. VCP can not distribute the bandwidth fairly between the flows that have huge RTT variation (with throughput ratio of up to 5). However, the bottleneck remains highly utilized, the persistent queue is very low and no packet gets dropped.

D. Dynamics

All the previous simulations focus on the long-term, steady-state behavior of VCP. Now we investigate its short-term dynamics.

Convergence Behavior: In this simulation, a single 45Mbps bottleneck is shared by 5 FTP flows that have 40ms, 50ms, 60ms, 70ms and 80ms RTT, respectively. There are also 5 FTP flows in the reverse direction. The forward flows start sending packets 100s apart at 0s, 100s, 200s, 300s and 400s, respectively. The reverse flows are always on. Figure 14 illustrates that VCP's AIMD mechanism reallocates bandwidth to new flows whenever they come in without affecting its high utilization or causing large instantaneous queue. Note these flows have different RTTs. The RTT heterogeneity is effectively compensated by the parameter scaling in Section III-C — otherwise, the flow with the smallest RTT will claim more bandwidth than the ones with larger RTTs. Finally, due to the reason discussed in Section III-D, VCP's fairness convergence takes much longer time than XCP. This is the price that an end-host based scheme without using precise bandwidth information has to pay.

Sudden Demand Change: In this simulation, traffic demands change suddenly and considerably. We start the simulation with 10 FTP flows sharing an 100Mbps bottleneck. They have RTT from 60ms to 105ms, 5ms apart. There are 10 FTP flows on the reverse path as well. At 40s, 40 new forward FTP flows become active. Then they leave at 80s. Figure 15 clearly shows that, at 80s when the departure of four fifth of the total traffic creates four-fold available bandwidth for the remaining flows, the system quickly discovers this and ramps up from 25% utilization to 95% in about 4 seconds! Notice that during the adjustment period, even though the bottleneck queue is more bursty, it remains very low. This simulation shows that VCP is responsive to sudden, significant decreases/increases in the available bandwidth. This is no surprise because VCP switches to the MI/MD mode which by nature can trace any bandwidth change Δbw in logarithmic time $O(\log(\Delta bw))$.



(1) Flow Throughput (2) Bottleneck Queue (3) Weight Achieved

Fig. 16. VCP is able to allocate differential bandwidth among competing flows of heterogeneous RTTs. Flow #1, #2 and #3 have 60ms, 80ms, 100ms RTT and a weight of 3, 2, 1, respectively. They share a 10Mbps bottleneck. (1) Bandwidth is allocated among three flows accordingly to their weights; (2) The bottleneck queue remains low all the time. (3) This separate set of simulations evaluate the achievable weight range. Among 10 flows of different RTTs in [60ms, 150ms], one of them (either flow #1 with 60ms RTT, or flow #5 with 100ms RTT, or flow #10 with 150ms RTT) has a weight w varying from 1 to 20 and all the others have a unit weight. The specified weights are all quite accurately achieved for flow #1, but the highest weights that flows #5 and #10 obtain are only 15.2 and 6.6, respectively, indicating the effect of their larger RTTs.

E. *Bandwidth Differentiation*

Finally we show VCP's bandwidth differentiation capability. In our first simulation, a 10Mbps bottleneck is shared by 3 FTP flows with different RTTs: $rtt_1 = 60$ms, $rtt_2 = 80$ms, $rtt_3 = 100$ms, and different weights: $w_1 = 3$, $w_2 = 2$, $w_3 = 1$. They all start at 0s but stop at 40s, 80s, and 120s, respectively. There are also 3 reverse FTP flows that are always on and all with weight 1. Figures 16(1)-(2) clearly demonstrate that the bottleneck bandwidth is distributed among the three flows according to their specified weights without introducing large queue in the bottleneck.

In our second set of simulations, we evaluate the range of weights achievable. We have a bottleneck of capacity 100Mbps shared by 10 FTP flows of heterogeneous RTT ranging from 60ms to 150ms ($rtt_i = 50 + 10 \times i$ ms for $i = 1, 2, \ldots, 10$). One flow has a varying weight in $[1, 20]$ while others have the same unit weight. There are also 10 reverse FTP flows all with unit weight. We simulate three cases with the weighted flow being flow #1 (60ms RTT), #5 (100ms RTT) and #10 (150ms RTT), respectively. Each simulation runs for 150s. Figure 16(3) shows the achieved bandwidth ratio between the weighted flow and the average of all the others. The achieved weight range for flow #1 is [1.0, 20.4], closely matching the specified wights, while flows #5 and #10 achieve only [1.0, 15.2] and [1.0, 6.6], respectively, indicating the influence of their larger RTTs. All these results are achieved while the bottleneck is highly utilized with low persistent queue.

RTT heterogeneity does limit the bandwidth differentiation achievability, as shown in Figure 16(3). This is due to the relationship between bandwidth differentiation and burstiness thus introduced into the network. The resulting burstiness will in turn affect if the targeted bandwidth differentiation is achieved or not.

V. Discussions

Since VCP switches its control between MI/AI/MD algorithms based on the load factor feedback, concerns naturally arise on the effect of mode sliding on system stability and bandwidth allocation efficiency and fairness, particularly when RTT heterogeneity is significant. In this section we discuss the efficiency, fairness and stability issues and comment on the incremental deployment aspects of VCP.

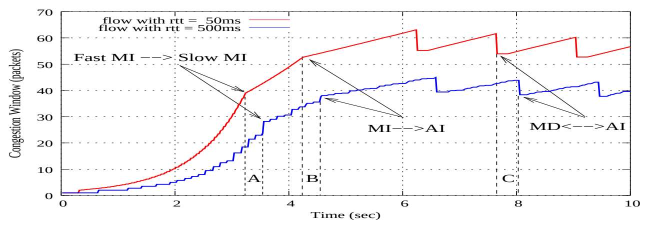


Fig. 17. The congestion window dynamics of two flows with dramatically different RTTs (50ms vs. 500ms). Due to its longer delay, the larger-RTT flow always slides its mode later than the small-RTT one (see the regions labeled as A, B, C). However, the effect of this asynchronous switching is limited by VCP and does not prevent it from achieving efficiency and fairness.

A. *Influence of Mode Sliding on Efficiency and Fairness*

From an efficiency perspective, VCP's goal is to bring and maintain the system to a high utilization regime and restrict mode switching between AI and MD. While MI enables VCP to quickly reach high link-utilization, VCP has several built-in mechanisms to enable the system to remain in that state. One such important mechanism is the scaling of AI/MI parameters in the case of flows having different RTTs. While one may argue that this scaling treats flows alike irrespective of the their RTTs which may seem undesirable, this behavior is critical to avoid oscillations. Specifically, this scaling is essential to prevent a flow with a very low RTT to apply MI several times within the same router load factor measurement period. Other mechanisms that VCP uses to maintain high efficiency include choosing an appropriate value of the MD parameter to remain in the high utilization regime, the safety margin between MI and AI, and applying burstiness limiters.

For fairness, as discussed in Section III-C, there are two major concerns: (1) Since RTT is the inherent control cycle of window-based schemes, small-RTT flows probe bandwidth faster than large-RTT flows; (2) A flow with higher RTT switches from MI to AI later than its lower-RTT counterparts and thus may gain an unfair share. The first issue is handled by the RTT scaling in Equations (13)-(16). The second issue is addressed by the MI burstiness limiter σ_{mi} (in the source host algorithm S.1). To further illustrate this, Figure 17 shows the congestion window evolution of two flows with RTT of 50ms and 500ms traversing a single 10Mbps bottleneck.

At 4.26s, the 50ms-RTT flow switches from MI to AI; the 500ms-RTT flow, however, keeps doing MI until 4.58s due to its longer delay. During this 0.32s interval, the 500ms-RTT flow's congestion window gain due to MI is limited by σ_{mi}. After that, it finally enters the AI phase which lasts for many rounds. The extensive simulation results in Sections IV-B $\sim$ IV-E, where we always use heterogeneous RTT settings, validate that VCP limits the effect of asynchronous mode-switching due to delay difference.

B. VCP Stability

We consider the stability of VCP based on the simplified model in Section II-C. VCP is a variable-structure system with two components: MIMD and AIMD. We have to consider both components as well as the mode sliding between them. The stability of AIMD has been well-established by [9] and [19]. Therefore we focus on MIMD and the sliding function.

Consider a topology of one bottleneck of capacity c used by N flows $i \in [1, N]$ with identical, constant RTTs. Assume $\forall i$, $rtt_i = T$. The source host i has a congestion window $w_i(t)$ at any time t. As shown in Figure 1, whenever a load factor $\rho(t)$ arrives at the source, due to the delay it was calculated with the aggregated source sending rate at time $t - T$:

$$\rho(t) = \frac{\sum w_i(t-T)}{cT}. \tag{32}$$

It triggers an update on the MI parameter $\xi(t)$ in Equation (3). Due to the MIMD rule in Equation (2), we have

$$\begin{aligned} \dot{w}_i(t) &= \frac{w_i(t)}{T} \cdot \xi(t) \\ &= \kappa \cdot \frac{w_i(t)}{T} \cdot [\frac{cT}{\sum w_i(t-T)} - 1]. \end{aligned} \tag{33}$$

Let $y(t) = \frac{cT}{\sum w_i(t)}$ (obviously $y(t) > 0$), we get

$$\dot{y}(t) = \frac{\kappa}{T} \cdot y(t) \cdot [1 - y(t-T)]. \tag{34}$$

It is called delayed logistic differential equation. Its stability has been well-understood [32]: If $\kappa \leq 3/2$, then it is globally asymptotically stable with the equilibrium $y^* = 1$.

We now discuss the mode sliding between MI, AI and MD. Under normal conditions, the system mode slides as MI $\rightarrow$ AI $\leftrightarrow$ MD. If somehow the system switches from MD directly to MI, thanks to the scaling of the MI/AI parameters in Section III-C, if there is no significant change in traffic demand, it will then slide into AI regardless of the length of RTT. Further, to prevent the system from oscillating between MI and MD, we use $\hat{\rho}_l = 0.8$ to separate MI and AI, and set the MD parameter $\beta = 0.875$. This way, a safety margin of 7.5% is provided.

Obviously, formal analysis on the mode-sliding function is still an open question. So, in this paper we rely on the extensive simulations in Section IV to evaluate its influence on the stability of VCP, which demonstrate substantially positive results, even for very high bandwidths (up to 80Mbps per flow) or large delays (up to 1.4s RTT). We also remark that TCP is also a variable-structure control — its slow-start is MI. VCP's load factor-guided MI is at most as fast as TCP slow-start.

C. Making VCP TCP-friendly

We define a VCP flow to be *TCP-friendly* with a competing TCP flow if the steady state throughput of the TCP flow matches what it would when competing with a normal TCP flow [42][13]. However in high BDP networks, a VCP flow should be able to leverage the additional bandwidth unused by the TCP flows while not affecting the throughput of TCP flows. Because VCP operates with AIMD in steady state, it is straight-forward to tailor it to exhibit TCP-friendly behavior. At the end host, to match TCP's AI parameter, we need to change the VCP AI parameter to $\alpha = \frac{3(1-\beta)}{1+\beta} = 0.2$ according to the TCP-friendly general AIMD formula [50]. At the router, when the encoded load factor $\hat{\rho}_l^c = (11)_2$, we replace the original deterministic ECN marking with a probabilistic one similar to RED. For TCP sources, in accordance with the ECN proposal, the encoded load factors $(00)_2, (01)_2$ and $(10)_2$ correspond to no congestion, while $(11)_2$ to congestion.

D. Incremental Deployment

If VCP is to be gradually deployed on the Internet, the deployment could follow the similar path as CSFQ [47] and XCP on an island-by-island basis. Therefore, even though VCP looks simpler than XCP, the deployment cost is quite similar, *not* much less. The deployment, however, will still benefit from VCP's simplicity: It does not need a new field in the IP header; the needed two-bit space has been standardized for congestion control purposes by the current ECN proposal and VCP uses it in a way that is a natural generalization of ECN. From the end hosts perspective, VCP can be made TCP-friendly, as described earlier. On the network side, as we have shown, the VCP router is scalable in that it does not keep any per-flow state and its algorithm complexity is very low. This makes it deployable in high speed core networks. The traffic inside an VCP island will immediately enjoy VCP's capability of maintaining high utilization, low persistent queue and minimal packet drop. The cross traffic that passes an VCP island between two border routers will be mapped onto an VCP flow from the ingress router to the egress router. These border routers do need to keep per-VCP-flow state. However, since the VCP flow is aggregated from the passing micro-flows, this will not cause scalability problems.

VI. Related Work

This work builds upon a great body of related work, particularly XCP [27], TCP [19][40], AIMD [9] and ECN [45][46]. Congestion control is pioneered by TCP [19] and the AIMD algorithm [9][22]. The research on AQM starts from RED [15][4], followed by Blue [12], REM [2], PI controller [18], virtual queuing [17], AVQ [34], and CHOKe [43], etc. A nonlinear optimization framework provides these works a theoretic underpin [29][39][33][28]. Below we relate VCP to three categories of congestion control approaches.

Explicit rate based schemes: XCP regulates source sending rate with decoupled efficiency control and fairness control and achieves excellent performance. ATM ABR (e.g., see [21] and the references therein) previously proposes explicit rate control. VCP learns from these schemes. However, VCP is primarily an end-host based protocol. This key difference brings new design challenges not faced by XCP and thus VCP is not just a "two-bit" version of XCP. The link load factor is suggested as a congestion signal in [21], based on which VCP quantizes and encodes it for a more compact representation of the degree of congestion. QuickStart [20] occasionally uses a number of bits per packet to quickly ramp up source sending rate. VCP is complementary to QuickStart in that it constantly uses two bits per packet.

Congestion notification based schemes: For high BDP networks, according to [27], the performance gap between XCP and TCP+RED/CSFQ/REM/AVQ with one-bit ECN support seems large. VCP generalizes one-bit ECN and applies some ideas from the above schemes. For example, RED' EWMA queue-averaging idea, REM's match-rate-clear-buffer idea and AVQ's virtual-capacity idea obviously find themselves in VCP's load factor calculation in Equation (6). This paper demonstrates that the marginal performance gain from one-bit to two-bit ECN feedback could be significant. Two-bit ECN is also used to choose different decrease parameters for TCP in [11], which is very different from the way VCP uses. On the end-host side, the binomial control [3] improves TCP's dynamic performance by generalizing AIMD, while VCP goes even further to combine MIMD with AIMD.

Pure end-to-end schemes: Recently there have been many studies on the end-to-end congestion control for high BDP networks. HighSpeed TCP [14] extends the standard TCP by adaptively setting the increase/decrease parameters according to the congestion window size. H-TCP [38] employs an adaptive AIMD with its parameters set as functions of the elapsed time since the last congestion event. Adaptive TCP [31] also applies dynamic AIMD parameters with respect to the changing network conditions. STCP [30] changes to a fixed MIMD algorithm. FAST [24] uses queueing delay, like TCP Vegas [5], instead of packet loss, as its primary congestion signal and improves Vegas' Additive-Increase/Additive-Decrease (AIAD) algorithm with a proportional controller. BIC TCP [49] adds a binary search phase into the standard TCP to probe the available bandwidth in a logarithmic manner. TCP Westwood [7] enhances the loss-based congestion detector using more robust bandwidth estimation techniques. All these end-to-end schemes do not need explicit feedback. Therefore, it is hard for them to achieve *both* low persistent bottleneck queue and zero congestion-caused packet loss. VCP does need explicit two-bit ECN but is able to maintain low queue and almost zero loss. However, it is unclear whether these end-to-end schemes, if given AQM/ECN support from network, can achieve similar performance as VCP in high BDP networks.

Variable-structure control with sliding modes has a long history in control theory [48]. It is useful when a set of features are desired in a system but no single algorithm can provide all of them. In computer networking areas, it has been used to solve the traffic engineering problem in [35]. Our work can be viewed as an application of this idea to congestion control.

VII. Conclusions

This paper demonstrates the existence of a simple, low-complexity congestion control algorithm for high bandwidth-delay product networks called VCP. Based on extensive ns2 simulations, we show that VCP can largely approximates XCP's performance while requiring only two bits of congestion feedback from the network. Given that the two-bit network feedback can be embedded in the ECN bits in the IP header, VCP becomes an attractive solution since it does not require any IP header modifications.

A few questions remain unaddressed. First, how much additional benefit can one get using more bits of network feedback? Second, to what extent pure end-to-end congestion control mechanisms scale for high BDP networks? Finally, from a deployment perspective, what is the right type of congestion control for high BDP networks?

VIII. Acknowledgement

The authors are grateful for the insightful comments from Sally Floyd and K. K. Ramakrishnan, and for the helpful suggestions on VCP's stability analysis from Xinzhe Fan and Jianghai Hu. Our thanks to all of them.

References

[1] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. *IETF RFC 2581*, April 1999.
[2] S. Athuraliya, V. Li, S. Low, and Q. Yin. REM: Active Queue Management. *IEEE Network*, 15(3):48-53, May 2001.
[3] D. Bansal and H. Balakrishnan. Binomial Congestion Control Algorithms. *INFOCOM'01*, April 2001.
[4] B. Braden et al. Recommendations on Queue Management and Congestion Avoidance in the Internet. *IETF RFC 2309*, April 1998.
[5] L. Brakmo and L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465-1480, October 1995.
[6] H. Bullot and R. Les Cottrell. Evaluation of Advanced TCP Stacks on Fast Long-Distance Production Networks. Available at http://www.slac.stanford.edu/grp/scs/net/talk03/tcp-slac-nov03.pdf.
[7] C. Casetti, M. Gerla, S. Mascolo, M. Sansadidi, and R. Wang. TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks. *Wireless Networks Journal*, 8(5):467-479, September 2002.
[8] A. Charny, D. Clark, and R. Jain. Congestion Control with Explicit Rate Indication. *IEEE ICC'95*, June 1995.
[9] D. Chiu and R. Jain. Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks. *Journal of Computer Networks and ISDN*, 17(1):1-14, June 1989.
[10] M. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Trans. Networking*, 5(6):835-846, December 1997.
[11] A. Durresi, M. Sridharan, C. Liu, M. Goyal, and R. Jain. Multilevel Explicit Congestion Notification. *SCI'01*, July 2001.
[12] W. Feng, K. Shin, D. Kandlur, and D. Saha. The Blue Active Queue Management Algorithms. *UMich CSE-TR-387-99*, April 1999.
[13] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. *SIGCOMM'00*, August 2000.
[14] S. Floyd. HighSpeed TCP for Large Congestion Windows. *IETF RFC 3649*, December 2003.
[15] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM ToN*, 1(4):397-413, August 1993.
[16] S. Floyd and V. Paxson. Difficulties in Simulating the Internet. *IEEE/ACM Trans. Networking*, 9(4):392-403, August 2001.
[17] R. Gibbens and F. Kelly. Resource Pricing and the Evolution of Congestion Control. *Automatica*, 35:1969-1985, 1999.
[18] C. Hollot, V. Misra, D. Towlsey, and W. Gong. On Designing Improved Controllers for AQM Routers Supporting TCP Flows. *INFOCOM'01*, April 2001.
[19] V. Jacobson. Congestion Avoidance and Control. *SIGCOMM'88*, August 1988.
[20] A. Jain and S. Floyd. Quick-Start for TCP and IP. *IETF Internet Draft draft-amit-quick-start-02.txt*, October 2002.

[21] R. Jain, S. Kalyanaraman, and R. Viswanathan. The OSU Scheme for Congestion Avoidance in ATM Networks: Lessons Learnt and Extensions. *Performance Evaluation*, 31(1):67-88, November 1997.
[22] R. Jain, K. K. Ramakrishnan, and D. Chiu. Congestion Avoidance in Computer Networks with a Connectionless Network Layer. *DEC-TR-506*, August 1987.
[23] H. Jiang and C. Dovrolis. Passive Estimation of TCP Round-Trip Times. *ACM Computer Communications Review*, 32(3):75-88, July 2002.
[24] C. Jin, D. Wei, and S. Low. FAST TCP: Motivation, Architecture, Algorithms, Performance. *INFOCOM'04*, March 2004.
[25] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan. Dynamics of an Explicit Rate Allocation Algorithm for Available Bit-Rate (ABR) Service in ATM Networks. *Proceedings of the IFIP/IEEE Conference on Broadband Communications*, April 1996.
[26] S. Kalyanaraman, R. Jain, S. Fahmy, R. Goyal, and B. Vandalore. The ERICA Switch Algorithm for ABR Traffic Management in ATM Networks. *IEEE/ACM Trans. Networking*, 8(1), February 2000.
[27] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. *SIGCOMM'02*, August 2002.
[28] F. Kelly. Fairness and Stability of End-to-End Congestion Control. *European Journal of Control* 9:149-165, 2003.
[29] F. Kelly, A. Maulloo, and D. Tan. Rate Control in Communication Networks: Shadow Prices, Proportional Fairness and Stability. *Journal of the Operational Research Society*, 49:237-252, 1998.
[30] T. Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *Submitted*, December 2002.
[31] A. Kesselman and Y. Mansour. Adaptive TCP Flow Control. *PODC'03*.
[32] Y. Kuang. Delay Differential Equations with Applications in Population Dynamics. Academic Press, 1993.
[33] S. Kunniyur and R. Srikant. End-To-End Congestion Control: Utility Functions, Random Losses and ECN Marks. *INFOCOM'00*, March 2000.
[34] S. Kunniyur and R. Srikant. Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management. *SIGCOMM'01*, August 2001.
[35] C. Lagoa, H. Che and B. Movsichoff. Adaptive Control Algorithms for Decentralized Optimal Traffic Engineering in the Internet. *IEEE/ACM Trans. Networking*, 12(3):415-428, June 2004.
[36] T. Lakshman and U. Madhow. The Performance of TCP/IP for Networks with High Bandwidth-delay Products and Random Loss. *IEEE/ACM Trans. Networking*, 5(3):336-350, June 1997.
[37] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the Self-Similar Nature of Ethernet Traffic. *SIGCOMM'93*, August 1993.
[38] D. Leith and R. Shorten. H-TCP: TCP for High-speed and Long-distance Networks. *PFLDnet'04*, February 2004.
[39] S. Low and D. Lapsley. Optimization Flow Control, I: Basic Algorithm and Convergence. *IEEE/ACM Trans. Networking*, 7(6):861-875, December 1999.
[40] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgement Options. *IETF RFC 2018*, October 1996.
[41] Network Simulator ns-2. Http://www.isi.edu/nsnam/ns/.
[42] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. *SIGCOMM'98*, September 1998.
[43] R. Pan, K. Psounis, and B. Prabhakar. CHOKe, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation. *INFOCOM'00*, March 2000.
[44] V. Paxson and S. Flyod. Wide-Area Traffic: The Failure of Poisson Modeling. *SIGCOMM'94*, August 1994.
[45] K. K. Ramakrishnan and S. Floyd. The Addition of Explicit Congestion Notification (ECN) to IP. *IETF 3168*, September 2001.
[46] K. K. Ramakrishnan and R. Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *SIGCOMM'88,*August 1988.
[47] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. *SIGCOMM'98*, September 1998.
[48] V. Utkin. Variable Structure Systems with Sliding Modes. *IEEE Trans. Automatic Control*, 22(2):212-222, April 1977.
[49] L. Xu, K. Harfoush, and I. Rhee. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. *INFOCOM'04*, March 2004.
[50] Y. Yang and S. Lam. General AIMD Congestion Control. *ICNP'00*, November 2000.