Improving the Fairness and the Response Time of TCP-Vegas

Abstract— **Unfairness of the Internet has galvanized numerous studies toward fair allocation of bandwidth. Study of TCP-Vegas is one of them. TCP-Vegas, although not perfect, at least enables bandwidth allocation independent of propagation delay, which is radically different behavior from that of current Internet. In the current Internet, a long-delay connection usually receives less throughput than short-delay connections.**

Until now, two necessary conditions have been identified to make TCP-Vegas achieve fair allocation of bandwidth: correct estimation of propagation delay and finer control of window by adopting single threshold rather than two.

In this paper, we propose three more fixes to achieve fair bandwidth allocation. First, we provide a fix for packet size independence. Second, we provide a fix regarding the reference value in the control. Third, we provide a fix for reducing both the oscillation and the convergence delay. We argue that fixes of ours and those of previous researchers constitute the necessary and sufficient condition for fair allocation of bandwidth.

Keywords—**TCP, congestion control, TCP-Vegas, fairness, packet size.**

I. INTRODUCTION

TCP is one of the most widespread protocols in the Internet. TCP is a connection-oriented transport protocol that provides reliable delivery service using unreliable IP network. There are several versions of TCP such as Tahoe, Reno, and Vegas. Every TCP version except TCP-Vegas adopts congestion avoidance scheme described in [1] whose basic idea is slowly increasing the number of packets in the network until loss occurs and halving it down when loss occurs. Each version improves performance of previous ones by revising behavior during loss recovery. In contrast, TCP-Vegas tries to gain performance improvement more fundamentally in that it does not incur loss because it does not wait for loss in reducing the number of packets in the network.

The principle of TCP-Vegas is to have per-connection queue length constant. This allows fair bandwidth allocation as well as loss prevention. It is well known that connections get the same throughput if all the connection have the same per-connection queue length.[1] Loss is prevented with a finite amount of buffer if the number of connection is bounded. If the connections are too many or the buffer is not large enough, TCP-Vegas resorts to TCP-Reno-like lossy behavior, hence TCP-Vegas is always no worse than TCP-Reno.

We focus on the fairness of TCP-Vegas because its fairness is superior to that of Tahoe and Reno. Tahoe and Reno are versions of TCP currently dominating the Internet. Under TCP-Reno or Tahoe, a long-delay connection usually receives less throughput than short-delay connections.

However, even TCP-Vegas has been known to be unfair for two reasons [13][11]. First reason is that TCP-Vegas relies on correct measurement of propagation delay, which is usually not met. A connection that incorrectly measures the propagation delay steals more bandwidth. A connection gets the less correct measurement of propagation delay as it shares the link with the more other connections. There has been no effective way[2] to enforce correct measurement of propagation delay. In this paper, we assume that every connection has correct propagation delay measurement. This may be implemented by giving SYN packets higher queueing priority than other types of TCP packets.

Second reason is that TCP-Vegas actually does not try to make per-connection queue length constant. Instead, it tries to bound per-connection queue length within a region marked by two thresholds, α and β, which are usually set as 1 packet and 3 packets respectively. The known solution of this artifact is to have both thresholds have the same value, which is adopted in this paper.

This paper shows that Vegas is still unfair even when the above two problems are solved, because of its dependence on packet size[3]. In the current Vegas, the connection with bigger packet gets more bandwidth. We propose three fixes as a solution.

First, we propose to redefine the thresholds, α and β, which are set equal to each other in this paper, as a byte

[1] This is the single bottleneck case. For the multiple bottleneck case, this type of allocation can provide a proportional fairness, instead of max-min fairness.

[2] [11] argued that using single threshold instead of two may solve this problem by incurring oscillation, which is later disproved by [13]. [9] mentioned the possibility of expiring fixed-delay measurement without any further development.

[3] Precisely, we mean Path MTU(Maximum Transmission Unit) size when speaking of packet size. MTU is the largest allowable payload size for link level frame. Path MTU is the least MTU along the forward path. MTU ranges from 296B of SLIP to 65536B of Hyperchannel.

count instead of as a packet count. We call the byte count threshold as the *reference*. This fix constitutes the core of our solution. The remaining fixes deal with the performance of this first fix.

Our second fix is about choosing the value of the reference. The first fix alone exhibits a significant departure from the perfect fairness, if the reference is not large enough. We show that a higher reference improves the fairness. We can always make the bandwidth allocation fairer by increasing the reference. However, this comes at the price of an increase in both the convergence delay and the buffer requirement.

Our third fix is to reduce the needed reference value in achieving the same level of fairness. Specifically, we propose to upgrade the congestion detection formula by adding a half packet. This improves the fairness when used under the same reference value, in that it eliminates the preference of larger-packet connections in the above two fixes.

Our last fix is to reduce the convergence delay when the reference value is high. This is a departure from the traditional window control in that the window change speed is not necessarily one packet a round. The window may change either by multiple packets a round or by less than one packet a round. We argue that this does not cause any harm on the stability of the congestion control, by providing a proof.

The rest of this paper is organized as follows. We describe the problem in Section II. We describe the main fix in Section III. We show the effect of the value of the reference in Section IV. We show how to fully eliminate the packet size dependence by upgrading the congestion detection formula in Section V. In Section VI, we present a faster window control, which shortens the transient state without losing the stability. A summary of our work and future plans is given in Section VII.

II. Vegas Throughput is Dependent on Packet Size

We show that TCP-Vegas is unfair because connections with different packet sizes receive different throughput in the steady state.

Fig. 2 shows that connection with bigger packet gets more bandwidth. There are two connections sharing a bottleneck link under the topology of Fig. 1. One connection has Path MTU of 1500B(limited by Ethernet or PPP), while the other has Path MTU of 296B(limited by SLIP). For many graphs in this paper, the legend of the graph means the packet size, and the vertical axis has no unit because it is a ratio of two values of the same dimension.

To show the effect of the current subject, we eliminated

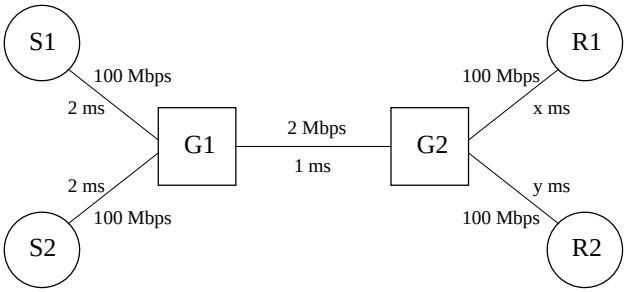

Fig. 1. Simulation Topology

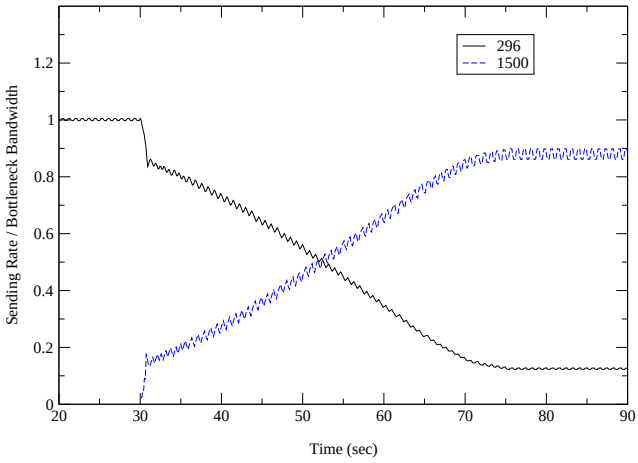

Fig. 2. Bigger Packet gets More Bandwidth

all the known causes of unfairness. We have made them know their correct propagation delay. We set two thresholds, α and β, equal to two packets.

III. Packet Size Independence

We can obtain much improved fairness by changing the semantic of the threshold from a packet count to a byte count. Fig. 3 is when the reference byte count is set to 3kB, which is chosen to be twice as high as the largest MTU in this simulation setting.

However, this improvement is still not perfect. We can observe that connection with bigger packet still gets slightly more bandwidth. We conducted a simulation of two connections, the Path MTU of one of them being changed, while that of the other having 296B Path MTU. The reference byte count was 3000 bytes. Figure 4 shows that the error is too high to be ignored and is increasing as the difference in packet size widens.

IV. Reducing the Error by Raising the Thresholds

In general, the fairness can be made better by raising the threshold.

When the reference byte count is raised to 131070 B,

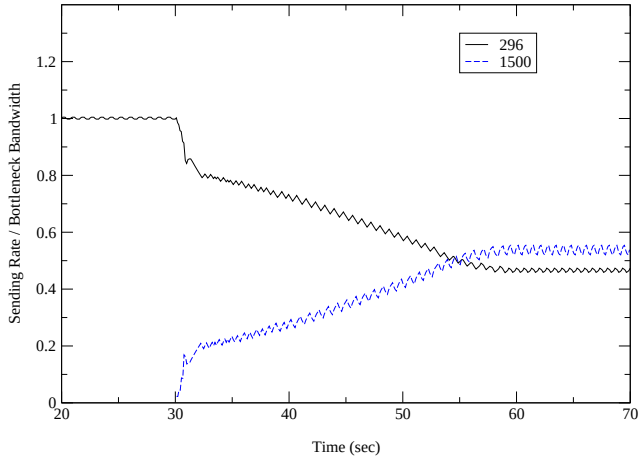

Fig. 3. Equalizing the Formula in Byte

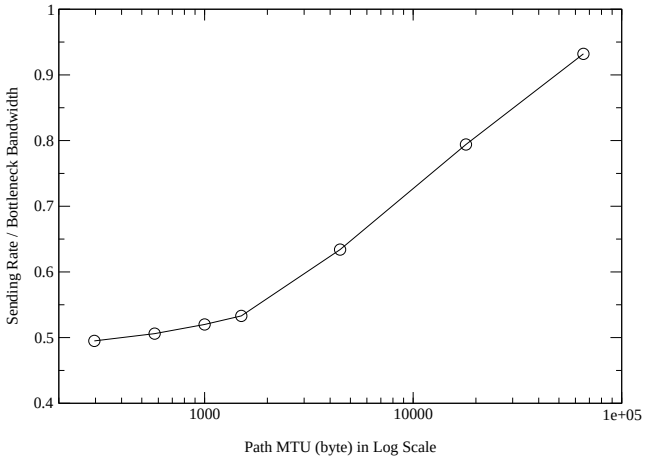

Fig. 4. Connection with Larger Packet Gets More Throughput When Competing with Connection with 296B Packet

which is chosen to be twice as the largest possible MTU, varying the difference in packet size does not cause difference of more than six percent in throughput.

However, there are two problems remaining in raising the reference.

The first problem is that the better treatment to connections with the larger MTU is all the same, though somewhat mitigated.

The second problem is that raising the reference count increases both the buffer requirement and the convergence delay. The increase in queue length is inferred by considering the reference byte count as the mean queue length. The increase in the convergence delay is shown in Fig. 6, where the steady state is reached only after 420 seconds.

The first problem is dealt with in the next section, and the second problem is dealt with in the section after the next one.

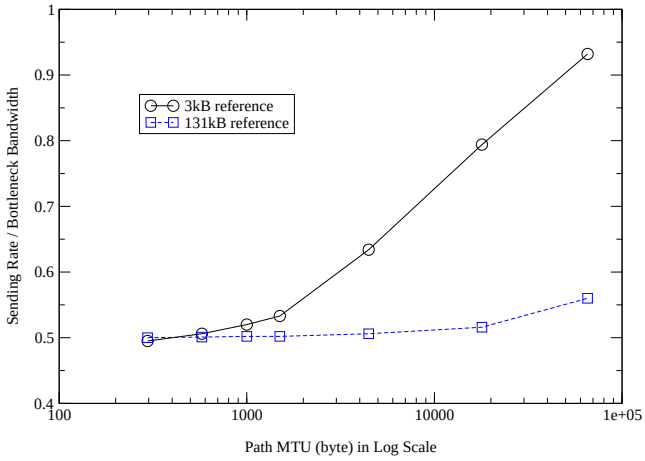

Fig. 5. Raising the Reference Byte Count Improves the Fairness

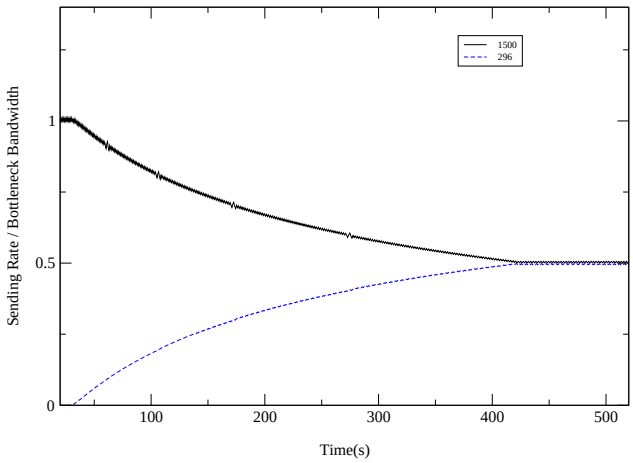

Fig. 6. Reference Byte Count Raised

V. Making the Error Independent of Packet Size

The results of the previous two sections suggest that the traditional interpretation of the congestion detection formula may be incorrect. The congestion detection formula has been known to mean the per-connection queue length. If the per-connection queue length is all the same throughout the connections, the connections should receive the same throughput. However, this is inconsistent with what the previous two sections have shown.

In this section, we show that the known interpretation of the congestion detection formula is indeed incorrect. Our proposed fix is simple and is shown to improve the fairness under the same reference byte count and to eliminate the preference of larger-packet connections, thus possibly enabling lowered byte reference count for the same level of fairness.

The fix is to add one half packet to the congestion detection formula. We argue that the resulting value means the

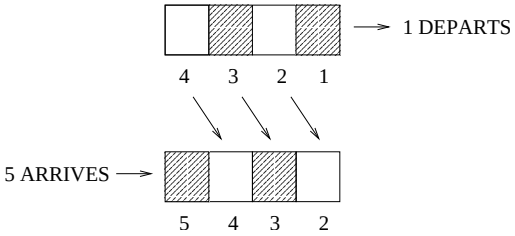

Fig. 7. Failure of Per-connection Queue Length Computation

per-connection queue length. This is because the packet arriving at the bottleneck queue fails to count itself when measuring queueing delay.

Figure 7 depicts this phenomenon when per-connection queue length is two. Just before a packet arrives, another packet of the same connection departs the bottleneck queue. So every packet sees only one packet in the queue ahead of itself, although there are always two packets in the queue.

There is yet another aspect remaining to be clarified: the figure says that we should add one packet, whereas our fix is to add a half packet. In fact, as shown by the proof in the appendix, the ideal fix is to add $\frac{P}{n}$, where P denotes the packet size, and n is the number of connections. The reason for not using this ideal fix is that it requires knowing the number of connections. Choosing a fixed n results in both the preference of small-packet connections when more than n connections share the bottleneck link and the preference of large-packet connections when less than n connections share it. We chose $n = 2$ because of the following two reasons: we want to make it impossible for the larger-packet connection to get more throughput; we want to make the queue length error as small as possible. The first reason suggests using either 1 or 2 as n, and the second reason dictates that $n = 1$ makes the error too large when the number of connections is large.

Fig. 8 shows that we can improve the fairness from Fig. 3 by adding a half packet even when using a low reference of 3kB.

This fix can also improve the fairness when using a higher reference count of 131kB, as shown in Fig. 9. This figure clearly shows that the fix makes the dependency on the packet size unobservable when the number of connections is two.

However, there is a shortcoming in this fix. It requires the reference count to be greater than half the largest possible MTU. Otherwise, the connection with the packet greater than twice the reference count would receive poor bandwidth, because adding a half packet would always result in a value above the reference, which in turn forces to decrease the window.

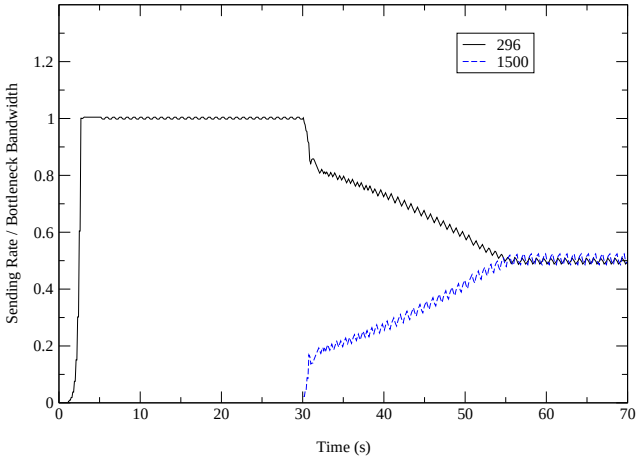

Fig. 8. Equalizing the Formula Plus One Half Packet

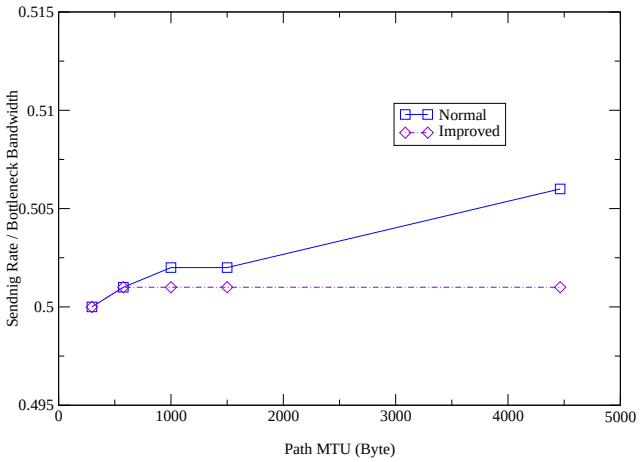

Fig. 9. Fairness Improvement by Adding a Half Packet(The Reference Count is 131kB)

VI. Reducing Both the Oscillation and the Convergence Delay

In this section, we present a fast window control scheme designed to solve the problem of long convergence delay, which is a consequence of raising the byte reference count.

Our fast window control is to make the rate of window change proportional to the error, which is defined as the difference between the reference and the estimate of the per-connection queue length. The normal Vegas also compares the estimate of the per-connection queue length with the reference. However, this is only used in determining the direction of change, and has no influence in the rate of the window change. The rate of the window change of the normal Vegas can assume one of the following three: plus one packet a round, minus one packet a round, or no window change during a round.

Our fast window control is sometimes slower than the normal Vegas, thus eliminating the oscillation.

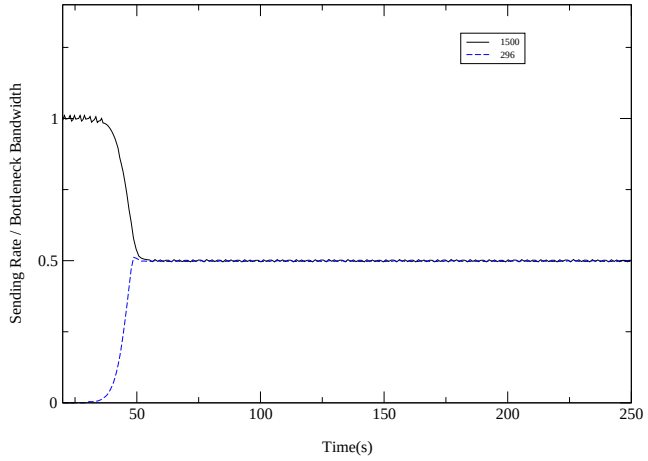

Fig. 10. After Using the Fast Control from Fig. 6

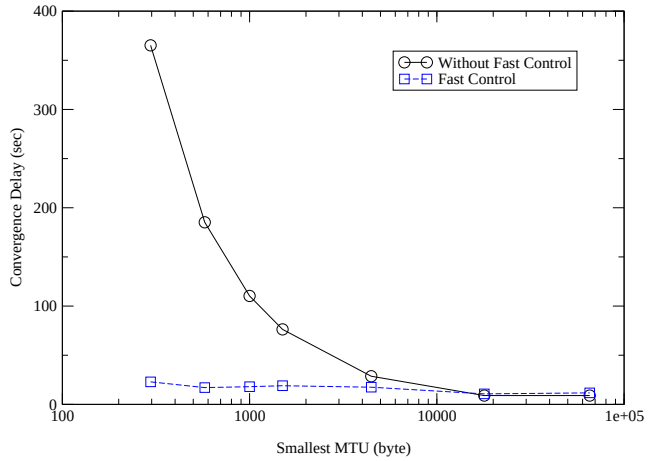

Fig. 11. The Fast Control Reduces the Convergence Delay

Figure 10 shows the application of our fast control on Fig. 6. The fast control reduces the convergence delay from 390 seconds to 30 seconds.

Figure 11 shows how much the convergence delay is reduced by our fast control. The simulation was carried out with two connections, where a connection with a controlled packet size competes with the other whose packet size is 65535B. The result indicates that, without fast control, the convergence delay is indirectly proportional to the smallest MTU, and that the fast control makes the convergence delay almost independent of the MTU size.

We argue that our fast control scheme is stable, based on the following argument: the fast control is stable if the normal Vegas is stable regardless of the MTU size. A small-packet connection changing multiple packets of window a round is not different from a large-packet connection changing one packet a round.

VII. Conclusions

Vegas shows much improved fairness than Reno or Tahoe. Its bandwidth allocation is independent of the propagation delay, whereas Reno or Tahoe give less throughput to long delay connections.

This improved fairness results from its principle of congestion control: it tries to keep constant number of packets in the bottleneck queue. In theory, this principle implies the perfect fairness in throughput.

However, Vegas does not show perfect fairness, because of some implementation problems. Two such problems have been identified by previous studies: it does not always obtain correct measurement of the propagation delay; it does not actually equate the per-connection queue length among all the connections.

This paper explains another problem resulting from different path MTU sizes among connections. To solve this problem, this paper proposed one main fix and three subsidiary fixes. The main fix is that the Vegas should try to keep constant number of bytes, rather than packets, in the bottleneck queue. The subsidiary fixes answer the following three questions: how many bytes should be kept in the bottleneck queue; whether the main fix is sufficient; what the side effect is and how to solve it.

In doing this, we found that the Vegas implementation has yet another discrepancy from its principle, in an aspect other than the data unit being packet or byte. We found that the measurement of the per-connection queue length is incorrect.

We also proposed an improved window control, which allows the rate of the window change to be much faster or much slower than the conventional rate of one packet a round. A formal proof and a thorough validation on its stability is our future work.

References

[1] V. Jacobson, *Congestion Avoidance and Control*, Proceedings of ACM SIGCOMM'88, August 1988.

[2] S. Keshav, *A Control-Theoretic Approach to Flow Control*, Proceedings of the SIGCOMM '91, pp. 3-15, September 1991.

[3] Lawrence S. Brakmo and Larry L. Peterson, *TCP Vegas : End to End Congestion Avoidance on a Global Internet*, IEEE J. of Selected Areas in Communication, pp.1465-1480, October 1995.

[4] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, *Evaluation of TCP Vegas: Emulation and Experiment*, ACM SIGCOMM '95, Cambridge, MA, August 1995.

[5] J. S. Ahn and P. B. Danzig, *Packet Network Simulation: Speedup and Accuracy Versus Timing Granularity*, IEEE Transactions on Networking, 4(5):743-757, October 1996.

[6] J.C. Hoe, *Improving the Start-up Behavior of a Congestion Control Scheme for TCP*, Proceedings of ACM SIGCOMM'96, August 1996.

[7] Y. Zhang, E. Yan, and S.K. Dao, *A Measurement of TCP over*

Long-Delay Network, Proceedings of 6th Intl. Conf. on Telecommunication Systems, pages 498-504, March 1998.

[8] T. Bonald, *Comparison of TCP Reno and TCP Vegas via Fluid Approximation*, Workshop on TCP, 1998.

[9] Jeonghoon Mo, Richard J. La, Venkat Anantharam, and Jean Walrand, *Analysis and Comparison of TCP Reno and Vegas*, IEEE INFOCOM'99, pp.1556-1563, 1999.

[10] S. McCanne and S. Floyd, *Ns(network simulator)*, http://www-mash.cs.berkeley.edu/ns

[11] G. Hasegawa, M. Murata, and H. Miyahara, *Fairness and Stability of Congestion Control Mechanisms of TCP*, IEEE Infocom 1999.

[12] A. Aggarwal, S. Savage, and T. Anderson, *Understanding the Performance of TCP Pacing*, IEEE INFOCOM 2000, 2000.

[13] C. Boutremans and J. Le Bourdec, *A Note on the Fairness of TCP Vegas*, International Zurich Seminar on Broadband Communications, 2000.

[14] Jin-Ru Chen and Yaw-Chung Chen, *Vegas Plus: Improving the Service Fairness*, IEEE Communication Letters, Vol. 4, No. 5, May 2000.

[15] U. Hengartner, J. Boliger, and T. Gross, *TCP Vegas Revisited*, IEEE INFOCOM 2000.

APPENDIX

I. THE CONGESTION DETECTION FORMULA DOES NOT MEAN THE PER-CONNECTION QUEUE LENGTH

In this section, we explain why the sender under-estimates its per-connection queue length, and the motivation of our solution. The reason of the under-estimation is that the queueing delay measurement is under-estimated and that the queue length is only inferred by the measured queueing delay.

First, we need to show how the per-connection queue length is inferred from the queueing delay. This is done by the Vegas *congestion detection formula*. Suppose t_n is the beginning of the n-th round. At time t_n, the congestion detection formula is defined as follows:

$$Diff \triangleq \tau_q * \lambda, \tag{1}$$

where the queueing delay is $\tau_q \triangleq RTT(t_n) - baseRTT$, and the throughput is $\lambda \triangleq \frac{W(t_n)}{RTT(t_n)}$, W(t) is window size at time t, and $baseRTT$ is the least of Round-Trip Times (RTT) until then. We assume $baseRTT = \tau$, where τ is the round-trip propagation delay. $RTT(t_n)$ is the average of the round-trip times experienced by the acknowledgments received during $[t_{n-1}, t_n)$. The unit of queue length is that of window used in this formula.

Second, we show a formal argument about the inevitable under-estimation of the queueing delay. We have already tried to give an intuitive explanation of this phenomenon, in Fig. 7. However, readers may not feel assured. This is driven by the steady-state assumption. Under steady-state, the aggregate queue length is the same and the per-connection queue length is the same. Thus if a packet of size P arrives, a packet of the same size must have left just before. Hence the larger the packet, the less bytes of queue are ahead of it in the queue. From this reason, we can say that the queueing delay is more under-estimated with larger packet size. Formally, the measured queueing delay is a decreasing function of the connection i's packet size P_i: $\frac{Q_A - P_i}{C}$, where the aggregate queue length is $Q_A \triangleq \sum Q_i$, Q_i is the actual per-connection queue length of connection i, and the bottleneck bandwidth is C. Thus, the per-connection queue length of connection i is under-estimated by $\frac{\lambda}{C} P_i$, if we combine this and the last paragraphs. If the bandwidth is fairly allocated and there are n connections, this amount equals $\frac{P_i}{n}$, because $\lambda = \frac{C}{n}$.

Using the above argument, we can infer that each connection should consider its amount of under-estimation in choosing the target per-connection queue length, in order to have the same per-connection queue length for every connection. When the per-connection queue lengths are actually the same, it is natural that the connection with bigger packet sees lower estimated queue length. However, the current Vegas implementation tries to equate the estimated queue length rather than the actual queue length. Our proposal is that a connection with bigger packet should set its target queue length lower. More specifically, when there are n connections, connection i should set its target as $alpha - \frac{P_i}{n}$. However, the number of connections n is not available, thus a conservative approach is to set the target as $alpha - \frac{P_i}{2}$. With this scheme, no connection with bigger packet can steal the bandwidth.

This phenomenon is problematic when the packet size is heterogeneous. Note that our proposed scheme does not cause any harm when every packet size is the same.