HYBRID WINDOW AND RATE BASED CONGESTION CONTROL FOR DELAY SENSITIVE APPLICATIONS

Sanjeev Mehrotra #, Jin Li # Sudipta Sengupta #, Manish Jain *, Sayandeep Sen †,

# *Microsoft Research, Redmond, WA, USA* – {sanjeevm,jinl,sudipta}@microsoft.com

* *Akamai, Boston, MA, USA* – jain.manish@gmail.com

† *Univ. of Wisconsin, Madison, WI, USA* – sayandeeps@gmail.com

***Abstract*—There has been a dramatic increase in interactive cloud based software applications. Compared to classical real-time media applications (voice over IP (VoIP)/conferencing) and non real-time file delivery, these interactive software applications have unique characteristics: 1) they are delay sensitive yet demand in order and reliable data delivery, and 2) the traffic is usually bursty. Traditional window based congestion control does not work well for interactive applications because the bursty arrival of data leads to bursty network traffic, which causes additional queuing delay and packet loss in the network which reduces its delay performance. In this paper, we propose a new hybrid window plus rate based congestion control technique. This algorithm improves the delay performance of interactive applications by preventing congestion induced loss and minimizing queuing delay while still fully utilizing network capacity and maintaining fairness across multiple flows.**

I. INTRODUCTION

Online interactive software applications are flourishing. For example, most web pages are no longer static and require constant interaction (e.g. web based e-mail, financial websites). Another example is online multi-player games such as World of Warcraft and Final Fantasy XI. A further example is software as a service (SAAS) such as Google Apps and Microsoft Office Web Apps, where massive scalable IT-enabled capabilities are delivered to customers. SAAS is projected to grow to $15 billion by 2012[4], increasing its share in the enterprise software market from 10.7% in 2007 to 18.2% in 2012. One crucial aspect that affects the user experience of an interactive application is its responsiveness. As an example, consider an application where a thin client is used for display and input (keyboard/mouse) purposes and the server is located in a distant data center. The server processes the incoming commands and the application responds by providing a screen update to the client. The responsiveness of the application is directly related to the timely delivery of this response. In addition the server response traffic is bursty in nature.

Since most interactive applications operate as a state machine, the data has to be delivered losslessly and in-order so that the client and server state are in sync. Therefore most existing applications simply use TCP (New Reno) [2]. However the use of TCP for interactive applications suffers from two main issues: 1) the use of only retransmissions for reliability results in large delays whenever there is loss (congestion or otherwise) and 2) the use of a window based congestion control and only reducing rate in presence of packet loss can result in large queuing delays and packet loss in congested cases (whenever the instantaneous (burst) sending rate of the application is larger than available bandwidth of the bottleneck link). Since packet loss may result in retransmission (even with the use of forward error correction (FEC)), both network queuing delay and packet loss can lead to poor delay performance for interactive applications.

In this paper, we aim to build a low delay reliable data transmission protocol on top of user datagram protocol (UDP) as an alternative to TCP. We have investigated the use of forward error correction (FEC) codes to reduce the delay caused by retransmissions in [3], [7]. In this paper, we turn our attention to the congestion control portion of the protocol. Our goal is to develop a protocol which minimizes queuing delay and packet loss while fully utilizing link capacity and maintaining fairness across flows. Two things are needed to ensure that queuing delays and packet losses are minimized: 1) ensure that the instantaneous sending rate is close to or below the actual available bandwidth so that queuing delay does not build up, and 2) use queuing delay as an indicator of congestion and use a congestion detection threshold which is close to the desired queuing delay level.

TCP uses window based additive-increase, multiplicative-decrease (AIMD) schemes to guarantee fairness across multiple flows and maintain full link utilization [1]. However, the use of a window can result in large network queuing delays for bursty traffic (as large as RTT) and congestion induced packet loss. For full link utilization, the window should be equal to the flow's share of bandwidth times the round trip time (RTT). Thus although the average sending rate is close to the bandwidth, the instantaneous sending rate can be much higher – since an entire window of data can be pushed out at once resulting in large queuing delays. Even delay based schemes such as TCP Vegas/FAST TCP may still suffer from similar issues for bursty traffic since they are also window based. One apparent solution to this is to pace the packets rather than allowing a full window of packets to go out at once. In this paper, we accomplish this by designing a novel rate plus window based rate control technique.

Although the use of pacing allows us to perform congestion detection by looking at queuing delay with a congestion detection threshold which is less than RTT, the use of a low threshold can result in link under-utilization. It is known that if TCP New Reno is used on high bandwidth-delay product (BDP) networks, then if the buffers are smaller than BDP, the link is not fully utilized since the congestion detection threshold – which for TCP New Reno is the buffer size since only loss implies congestion – is smaller than the BDP. Many solutions have been proposed to improve link utilization such as [6], [9], [8] using faster ramp-up. Our issue is similar since we use a desired queuing delay (smaller than BDP) as the congestion detection threshold. Thus our solution adopts a similar strategy.

In this paper, we present a novel congestion control technique using a hybrid rate plus window based rate control to minimize queuing delay and packet loss. It uses AIMD to guarantee fairness across multiple flows but uses fast ramp-up and graceful back-off to prevent link underutilization caused by lower congestion detection thresholds. The use of pacing and lower congestion detection thresholds allows us to control queuing delay to desired levels. Also since our congestion detection thresholds are less than RTT – and thus with very high probability less than buffer sizes – we can ignore packet losses which are not accompanied by a delay increase. This improves the performance of our protocol on links with random losses, such as wireless links.

II. Congestion Control Protocol

The goal of a congestion control protocol is to control the transmission rate so that packets suffer minimal network queuing delay and loss caused by congestion while sustaining throughput close to the available network bandwidth and ensure fair sharing of network resources. Most congestion control protocols use an additive-increase multiplicative-decrease (AIMD) scheme to adjust the transmission rate (R) or window (W) at the nth time instant using

$$W_{n+1} = \begin{cases} W_n + \alpha & \text{if no congestion} \\ W_n(1-\beta) & \text{if congestion} \end{cases}. \tag{1}$$

The details of any AIMD scheme is in its definitions of congestion, α (amount to increase), and β (amount to decrease). Congestion is defined in terms of loss, delay, and explicit congestion notification signals (if present). Since AIMD control schemes guarantee fairness regardless of network state [1], we use it as the base of our protocol. However, as opposed to a congestion control strategy used for large file transfers (e.g. TCP) which attempts to maximize throughput, the congestion control strategy employed in our protocol has to be sensitive to delay (minimize network queuing delay and packet loss).

Based on our requirements, the key characteristics of our protocol are: 1) hybrid rate plus window based rate control technique (to minimize queuing delay and packet loss), 2) fast ramp-up / graceful back-off (to prevent link under-utilization), 3) using AIMD (to provide fairness), and 4) delay based congestion detection (to achieve any desirable queuing delay level). The amount of ramp-up (additive-increase) and back-off (multiplicative-decrease) uses queuing delay as an indicator of level of congestion and is inspired by the work in [6] which shows that a convex ramp-up curve as in Fig. 1 is optimal.

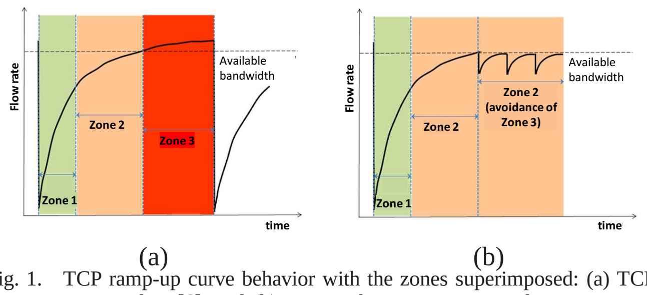

Fig. 1. TCP ramp-up curve behavior with the zones superimposed: (a) TCP congestion control in [6], and (b) Proposed congestion control.

Our protocol operates on epochs of length L which are defined to be units of time equal to the estimated round trip propagation time (the minimum RTT seen so far). At the end of each epoch, the epoch length is updated, the congestion level is classified into one of three zones as described in Sec. II-A, and the the transmission rate is updated as described in Sec. II-B. Rate control is performed as described in Sec. II-C.

A. Congestion Level Zone Classification

Instead of simply defining congestion as a binary event (congestion or no congestion), we use a continuous definition of congestion level using both packet loss and queuing delay. We first estimate the queuing delay (δ_{avg}) as the average of relative one-way delay (ROWD) measurements of all packets which have been acknowledged in the previous epoch. The ROWD is computed as $ROWD = OWD - OWD_{min}$, where OWD is the actual one-way delay computed as the received time using the receiver's clock minus the sent time using the sender's clock. OWD_{min} is the minimum OWD seen so far. Although OWD is sensitive to clock offset, $ROWD$ is not since OWD_{min} is an estimate of the propagation delay plus the clock offset. To prevent $ROWD$ measurements from being sensitive to clock drift (where one clock is running faster than the other), OWD_{min} can be taken to be the minimum over some window of measurements rather than the actual minimum. We then classify the congestion level into one of the following three zones.

- **Zone 1:** OWD trend is non-increasing and average queuing delay is less than some threshold ($\delta_{avg} \leq d_1$).
- **Zone 2:** OWD trend is non-increasing, no packet is lost, and $d_1 < \delta_{avg} \leq d_2$, for $d_2 > d_1$.
- **Zone 3:** OWD trend is increasing, $\delta_{avg} > d_2$, or packet loss is accompanied by a delay increase ($\delta_{avg} > d_1$).

To compute the OWD trend, we use the method developed in [5]. If packets are being properly paced, an increasing OWD trend means buffers are building up and thus implies congestion. The use of delay in determining congestion level is a commonly used technique. However, our classification of congestion level into three zones is unique, as is the use of OWD trend as an additional indicator of congestion.

The zone classification has an intuitive explanation. Ideally, we would always like to stay in Zone 2 (the zone which gives

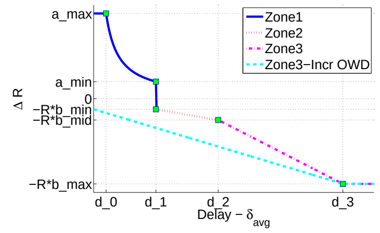

Fig. 2. Rate update based on zone (showing window change $\Delta R = R_{n+1} - R_n$). Two curves are shown - one is for non-increasing delay trend, other is for zone 3 with increasing delay trend. In the figure, “b” stands for β and “a” stands for α.

the tolerable queuing delay) and thus the typical queuing delay seen is between d_1 and d_2. Going to Zone 1 results in a queuing delay below what we want and lower link utilization. Zone 3 results in higher queuing delay and probability of congestion induced loss. Thus delays larger than d_2 will only be seen when new flows enter. By appropriately choosing d_1 and d_2 and accounting for typical propagation delay seen on the link, the end-to-end delay due to the network can be controlled. We illustrate the three zones by superposing them on a typical (unmodified) TCP-Illinois flow bandwidth ramp-up curve (Fig. 1(a)). Our modifications attempt to avoid Zone 3 and are shown in Fig. 1(b).

Note that we do not back off due to packet loss unless it is accompanied by a delay increase. This allows us to not be as sensitive to random losses, such as caused by wireless links.

B. Rate Update

At the end of every epoch, the transmission rate (R) and window (W) are updated based on the congestion classification. Instead of updating the window, the transmission rate is directly updated using

$$R_{n+1} = \begin{cases} R_n + \alpha & \text{if Zone = Zone 1} \\ R_n(1-\beta) & \text{if Zone = Zone 2 or Zone 3} \end{cases}, \quad (2)$$

$$\alpha = \begin{cases} \alpha_{max} & \text{if } \delta_{avg} \le d_0 \\ \frac{\alpha_{min}\alpha_{max}(d_1-d_0)}{\alpha_{max}(\delta_{avg}-d_0)+\alpha_{min}(d_1-\delta_{avg})} & \text{else} \end{cases}, \quad (3)$$

$$\beta = \begin{cases} \beta_{min} + \frac{\beta_{mid}-\beta_{min}}{d_2-d_1}(\delta_{avg}-d_1) & \text{if Zone = Zone2,} \\ \beta_{min} + \frac{\beta_{max}-\beta_{mid}}{d_3-d_2}(\delta_{avg}-d_2) & \text{if Zone = Zone3 \& OWD non-increasing \& } \delta_{avg} \le d_3, \\ \beta_{min} + \frac{\beta_{max}-\beta_{min}}{d_3}\delta_{avg} & \text{if OWD increasing \& } \delta_{avg} \le d_3, \\ \beta_{max} & \text{if } \delta_{avg} > d_3 \end{cases}. \quad (4)$$

$\alpha = \alpha_{max}$ for $\delta_{avg} \le d_0$ and decays to $\alpha = \alpha_{min}$ by the Zone 1 boundary d_1. The above β is used in case of no packet loss. β_{min}, β_{mid}, and β_{max} are used to control the shape of the β curve. β goes from β_{min} to β_{mid} during Zone 2, and then up to β_{max} in Zone 3 if the delay trend is non-increasing. If the delay trend is increasing, then it is assumed to be a sign of congestion and β linearly increases as a function of delay up to β_{max} regardless of queuing delay. For cases where packet loss is encountered and $\delta_{avg} > d_1$, $\beta = \beta_{max}$.

This rate update is illustrated in Fig. 2 for both cases when delay trend is increasing and when it is non-increasing. In Fig. 1(b), we show how our modification avoids the original sharp drop in sending rate in Zone 3 and how the new ramp-up curve looks. Instead of increasing in Zone 2, we decrease the rate in Zone 2, but not as aggressively as in Zone 3. The rate update has the property of fast ramp up / graceful back-off (since it is inspired by the work in [6]) and fairness across multiple flows (since it is AIMD). The fast ramp-up prevents link under-utilization which can occur if the thresholds d_1 and d_2 in Sec. II-A are smaller than RTT.

C. Window vs. rate based congestion control

TCP uses window based congestion control, in which the window size defines the maximum number of bits that can be outstanding. The protocol is allowed to transmit a packet so long as the number of outstanding bits (call it F) is less than the window size (W). The outstanding bit count increases whenever a new packet is sent and reduces once the packet is acknowledged (ACK) or once the packet times out (NACK). However, in media streaming applications, rate based congestion control is frequently used. In such applications, the application controls the transmission rate directly. The sender is allowed to send packets at the rate of R bits/second, regardless of the outstanding bit count.

The main advantage of window based congestion control is its self-clocking behavior since the sender is not able to increase the sending rate too fast if packets are suffering a large queuing delay (since the outstanding bit count only reduces on ACK or NACK). Window based congestion control can send out a burst of packets for bursty applications. Although for small bursts this can result in packets potentially having a lower end-to-end delay since they do not incur pacing delay, for a large burst of packets, some packets can experience a large queuing delay and even packet loss since the instantaneous sending rate can be much larger than the average. In our congestion level classification, this can occur if the thresholds d_1 and d_2 are smaller than the RTT.

Thus in our protocol, we further combine window based congestion control with a rate based congestion control scheme. That is, we use a window to control the maximum number of outstanding bits, but also control the rate at which packets can enter the network using a transmission rate. The transmission rate R (in bits/sec) is the quantity which is directly adjusted based on congestion signals and a window of size $W = RL$ (in bits) is used to control the maximum number of outstanding bits, where L is the epoch length.

In a pure rate based scheme, if we send a packet of size P bits, then with a transmission rate of R, we are only allowed to send the next packet after P/R seconds. In a pure window based scheme, we are allowed to send immediately so long as $F < W$. In our scheme, we want to pace the packets but at the same time not exceed the window and thus use a joint scheme. Suppose packet l of size P_l bits is sent at time $t = T_l$, then we are allowed to send the next packet ($l+1$) of size P_{l+1} at time t so long as $t > T_l + \gamma P_l/R$ and if $F < W$, where $\gamma \in [0.0, 1.0]$ is the pacing factor. When $\gamma = 1.0$, the congestion control is fully paced and is a joint rate based control with a window.

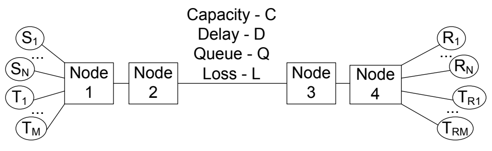

Fig. 3. Simulation Topology.

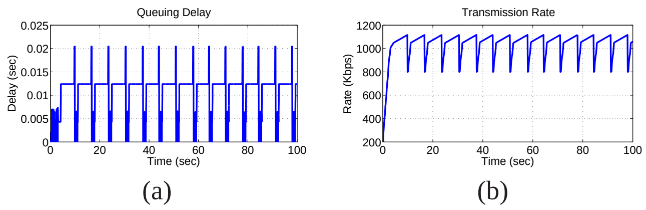

Fig. 4. Results using our protocol (single flow), (a) Queuing delay (δ_{avg}) (b) Transmission rate (R).

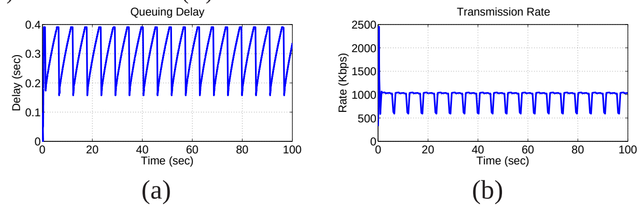

Fig. 5. Results using TCP New Reno.

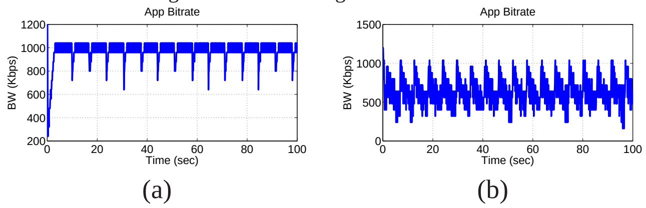

Fig. 6. Application bitrate (a) Using our protocol (b) Using TCP New Reno.

If $\gamma = 0.0$, it reverts to the simple window based rate control as in TCP. Once a packet is sent at time t, the number of outstanding bits updates as $F \leftarrow F + P_{l+1}$ and the last sent time is updated $T_{l+1} = t$. Upon ACK or NACK (timeout) of packet m, the outstanding bit count is reduced, $F \leftarrow F - P_m$.

III. Experimental Results

In this section, we evaluate the performance of our congestion control protocol using the ns-2 network simulator. We construct a standard multi-hop dumbbell topology as shown in Fig. 3, where the middle link is the bottleneck link. The bottleneck link has a capacity C, delay D, a queue with buffer size Q, and potentially a loss rate of L. The queue at the bottleneck link is a droptail queue. There are N sources S_i with corresponding receivers R_i.

The source is assumed to be one which can generate periodic bursts of up to P packets with a gap of G seconds giving a source rate of S. It is assumed to have a rate control module which operates as follows. The source generates packets and puts them into a buffer of size B. This buffer empties packets once they are sequentially decodable (the packet and all previous packets have been acknowledged). The source uses this buffer to perform rate control by simply checking if the buffer is full. For every burst, the source only generates packets which can fit into this buffer without overflowing. This is a relatively simple model of a bursty source, but is good for analysis purposes (for example a source will usually not generate bursts periodically, but rather with varying intervals between bursts).

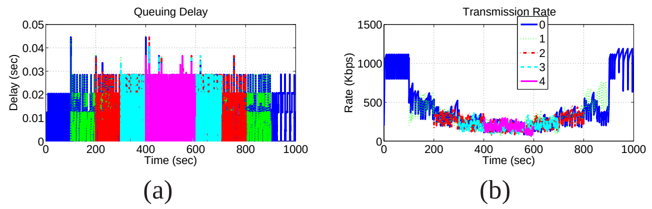

Fig. 7. Results using our protocol for five flows.

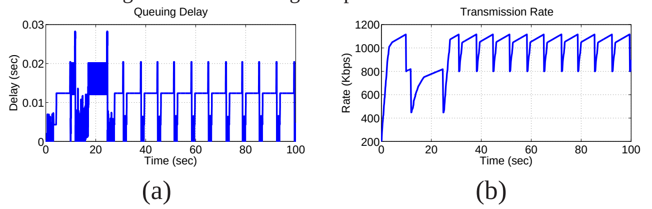

Fig. 8. Results when 300Kbps CBR cross traffic is introduced.

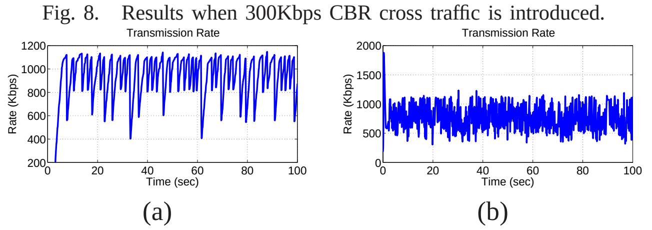

Fig. 9. Transmission Rate when random 5% loss is present. (a) Our protocol, (b) TCP New Reno.

We show via evaluation that our protocol results in no packet loss and very low queuing delay when compared to traditional congestion control protocols while maintaining close to full link utilization and fairness across multiple flows. In fact the queuing delay typically stays between the desired Zone 2 thresholds (between d_1 and d_2).

For our simulations for the sources we use $P = 15$packets, $G = 0.1$seconds, which gives a maximum source rate of $S = 1.2$Mbps. The capacity of the bottleneck link is $C = 1$Mbps, with a delay of $D = 50$ms, and droptail queue of Q=50 packets. Since $C < S$, the source application has to limit its sending rate due to congestion. The sender buffer size is assumed to be B=32 packets, or roughly 0.23 seconds of data. For the protocol, we use $\alpha_{min} = 800$bps, $\alpha_{max} = 40$Kbps, $\beta_{min} = 0.25$, $\beta_{mid} = 0.33$, $\beta_{max} = 0.5$, $d_0 = 0$, $d_1 = 12$ms, $d_2 = 24$ms, $d_3 = 48$ms, and $\gamma = 1.0$. These parameters are found by tuning for our queuing delay requirements.

A. Performance and Stability of Protocol

We first compare our congestion control protocol with TCP-New Reno. We show the overall performance of our protocol for a single network flow in Fig. 4, which includes both network queuing delay (δ_{avg}) and transmission rate (R_n) over time. From Fig. 4, we see that our protocol is able to achieve a queuing delay which is always between the desired target delay of 12ms and 24ms. As soon as it gets larger than d_2, we back off and our delay reduces to zero. The loss rate is zero for our case. We can also see that the link is close to full utilization since we quickly ramp up when we are far from congestion (i.e. when the queuing delay is close to zero) using the convex ramp up curve. The transmission rate is always between 800Kbps and 1Mbps.

From Fig. 5, we see that TCP New Reno does not perform as well. In fact the queuing delay is very large (close to 400ms), which is the size of the network buffer queue Q since it does not back off until loss is encountered. The transmission rate also oscillates much more (between 500Kbps and 1Mbps) since the back off is also constant ($\beta = 0.5$) instead of being a function of delay. In addition, there is some packet loss (close to 0.4%) which would cause some packets to suffer a larger delay caused by retransmissions.

As explained in Sec. I, just using TCP Vegas would not help since queuing delays of up to RTT would still be seen (in this case up to 100ms), whereas our protocol does not see queuing delays much larger than 24ms. TCP-Illinois would also suffer similar delays.

In Fig. 6, we show the bitrate as seen by the applications perspective using the source generation model described above. We see that our protocol is able to achieve a much smoother bitrate as seen from the application's perspective than using TCP New Reno (typically between 800-1000Kbps as opposed to 500-1000Kbps).

B. *Fairness of Protocol*

In Fig. 7, we show the performance of the protocol when multiple flows are present. Here we show five flows, flow 0 is present from 0-1000sec, flow 1 is present from 100-900sec, flow 2 from 200-800sec, flow 3 from 300-700sec, and flow 4 from 400-600sec.

From Fig. 7(a), we see that the low queuing delay is maintained regardless of the number of flows. As new flows enter, the queuing delay sees a slight increase as the protocol goes into Zone 3 for a while. However, this is immediately reduced. From Fig. 7(b), we see that the protocol is able to fairly divide the network bandwidth amongst all flows present while utilizing close to full link capacity. The first 100 seconds of Fig. 7(b) is identical to Fig. 4(b) since only one flow is present. When a second flow enters (time period 100-200 seconds), it takes about 30-40 seconds for the two flows to achieve almost the same share (about 500Kbps). As more flows enter, they each divide the bandwidth almost equally after a short time period. As flows exit, the remaining flows divide the bandwidth. Again, we suffer no packet losses.

C. *Effect of cross-traffic*

In Fig. 8, we show the effect of constant bit rate (CBR) traffic which does not back off. We show the result when a competing 300Kbps CBR source enters the bottleneck link from time period 10-25 seconds. Our protocol responds gracefully and reduces its transmission rate but still is able to take the remaining 700Kbps (on average). The queuing delay does not suffer either and no packet losses are encountered.

Of course if a more aggressive flow is present (one which does not back off or backs off at a higher congestion level), then our flow will yield as much bitrate as the other flow desires. For a CBR flow, it will yield whatever the CBR flow rate is. For other flows such as TCP file downloads where the bitrate has no real limit, our share reduces to practically zero. This is a common issue when flows with differing levels of aggressiveness are present. The only solution is to detect such situations and compete with the most aggressive flow present which may result in larger queuing delays or loss.

D. *Effect of non-congestion losses*

In Fig. 9, we show the effect when random losses such as those present on wireless links exist. We show the transmission rate achieved by our protocol and that achieved by TCP New Reno. Since we only respond to losses if they are accompanied by a delay increase, we are able to maintain a much higher average transmission rate (900Kbps vs. 750Kbps).

Again, we are able to do this since our congestion thresholds for queuing delay are with high probability much smaller than the buffer size, and thus we will always detect congestion prior to the buffers being full. The use of OWD trend acts as an additional safeguard to prevent congestion induced losses.

IV. CONCLUSION

In this paper, we have presented a hybrid window plus rate based congestion control protocol. By using pacing (joint window plus rate based rate control) along with a rate control which backs off upon seeing a queuing delay larger than the desired, we are able to achieve no packet losses and bounded queuing delays. Combining this with the fast ramp-up and back-off upon congestion, we are able to achieve full link utilization and fair allocation of network resources amongst multiple flows.

REFERENCES

[1] D. Chiu and R. Jain. Analysis of the increase/decrease algorithms for congestion avoidance in computer networks. *Journal of Computer Networks and ISDN systems*, 16(1):1 – 14, June 1989.

[2] S. Floyd and T. Henderson. *The NewReno Modification to TCP's Fast Recovery Algorithm*, Apr. 1999. RFC 2582.

[3] Y. Huang, S. Mehrotra, and J. Li. A hybrid FEC-ARQ protocol for low-delay lossless sequential data streaming. In *Proc. Int'l Conf. Multimedia and Expo*, pages 718–725. IEEE, June 2009.

[4] J. Humphreys. Worldwide virtual machine software 2008-2012 forecast. May, 2008, IDC.

[5] M. Jain and C. Dovrolis. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. *IEEE/ACM Trans. Networking*, 11:537–549, Aug. 2003.

[6] S. Liu, T. Basar, and R. Srikant. TCP-Illinois: A loss and delay-based congestion control algorithm for high-speed networks. In *Proceedings of the 1st international conference on performance evaluation methodolgies and tools*, Oct. 2006.

[7] S. Mehrotra, J. Li, and Y. Huang. Minimizing delay in lossless sequential data streaming. In *Proc. Int'l Conf. Multimedia and Expo*. IEEE, July 2010.

[8] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A compound TCP approach for high-speed and long distance networks. In *INFOCOM*, pages 1 – 12. IEEE, Apr. 2006.

[9] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Trans. Networking*, 14:1246 – 1259, Dec. 2006.