Abstract - In TCP (Transmission Control Protocol), congestion control as well as error recovery are implemented by a sliding window. The dynamics of TCP (specifically, a mismatch between the TCP window and the bandwidth-delay product of the network) can sometimes cause the network switches or routers to accumulate large queues, resulting in buffer overflows, reduced throughput, unfairness and underutilization. It is generally accepted that there is a limit as to how much control can be accomplished from the congestion control mechanisms in the end systems. Some mechanisms are thus needed in the intermediate network elements to complement the endpoint congestion avoidance mechanisms. Network layer enhancements such as scheduling mechanisms and packet drop policies have been proposed which are aimed at improving fairness and throughput of the competing endpoint applications. In this paper we describe a new TCP rate control scheme based on a simple recursive algorithm. The idea behind the algorithm is to match the network load to the available resources by modifying at an intermediate network element, the receiver’s advertised window in TCP acknowledgments returning to the source. The scheme can be implemented in a router or switch for bandwidth management and does not require knowledge of network delays or maintenance of per-flow state.

INTRODUCTION

The Transmission Control Protocol (TCP) [1,2] is the primarily protocol for congestion control in the Internet. TCP flow control is based on controlling the end-to-end data transmission window as a function of the congestion state of the network. An important attribute of TCP congestion control mechanisms is that they do not assume any support from the network for explicit signaling of congestion state. TCP infers the congestion state of the network from implicit signals - arrivals of acknowledgments (ACKs), timeouts, and receipts of duplicate ACKs.

TCP congestion control is window-based. The sender keeps a congestion window (cwnd) whose size limits the number of unacknowledged packets the sender can have outstanding in the network. Upon receiving ACKs for successfully transmitted data, the sender increases its transmission rate by incrementing the size of its congestion window. At some point in time, the transmission rate can eventually exceed the network’s capacity. When this happens, queues build up in the routers and overflow, causing packets to be dropped. TCP assumes that packet loss is due to congestion and reduces its congestion window (its transmission rate) upon detecting the loss. Reference [2] provides an excellent overview of the TCP protocol.

In a high-latency network environment, the window flow control mechanism of TCP may not be very effective because it relies on packet loss to signal congestion, instead of avoiding congestion and buffer overflow. The basic problem is that TCP does not communicate directly with the network elements to be given its optimal or assigned traffic rate. By the time the source starts decreasing its rate because of packet loss, the network has already been overly congested. This problem exists because the design of TCP only considers the flow control needs of the receiver. It does not consider the flow control needs of intermediate hops in the network. Overflow in the network itself would be detected by the sender through timeouts or through acknowledgment arrival patterns. This model has problems coping with shared multi-hop networks, where the cause of packet loss is within intermediate hops in the network.

Ideally, we would want the source to respond to congestion (in the network and at the destination) before it occurs rather than acting when it is too late. Thus, one method to make TCP sensitive to network congestion and buffer overflows is to allow network elements between a source and destination to modify the receiver’s advertised window in TCP acknowledgments returning to the source. Through this the network becomes an active (and not a passive) participant in controlling congestion and buffer overflows. Along these lines, various TCP receiver-window adjustment schemes have been proposed in the literature [3, 4, 5]. However, some of these schemes suffer from a clear understanding on how the control parameters should be selected for optimal performance, require accurate roundtrip delay estimates which can be difficult or impossible to obtain in IP networks, or entail too much per-flow state at access nodes, essentially requiring per-flow window tracking. A number of companies are currently marketing IP devices that have TCP window adjustment capabilities [6, 7, 8, 9]. Some of the devices adjust only the TCP receiver’s advertised window and others implement priority queuing in addition to changing the advertised window. It is presently unclear how these companies implement their TCP window adjustment because the techniques are considered as trade secrets.

PROPOSED TCP RATE CONTROL SCHEME

The objective of the TCP window control scheme described here is to match the sum of the windows of the active TCP connections sharing the buffer in a router to the effective network bandwidth-delay product, thus avoiding...
The TCP Rate Control Scheme

When multiple TCP connections share a common link, the proposed approach matches the aggregate window sizes of all active TCP flows to the bandwidth-delay product of the network while at the same time providing all the connections with similar feedback to achieve fairness (in window allocation). The feedback is carried by returning TCP acknowledgments in the receiver's advertised window field. If the current value in the receiver's advertised window field, which is set by the destination system, exceeds the feedback value computed in the router, the receiver's advertised window is marked down to the feedback value. The computed feedback bounds the TCP window maintained at the source in order to limit packet losses in the router.

The difficult task here is the design of the feedback function to be implemented at the router. It should be noted that the dynamics of the system depends heavily on this function. The function can be designed to provide all TCP connections with similar feedback, and as a result have them operate with equal windows.

Our approach is to estimate the target (router's) advertised window size as a function of the input/output rate mismatch at the router buffer. The computed "router's advertised window" is then used to mark down the receiver's advertised window field in the acknowledgments. Since setting the window size too small can lead to starvation, a minimum window can be enforced.

Let \( W_{NE}(0) \) be an initialized router's advertised window for a given traffic class (in bytes). Fix \( \Delta t \) as the period of the sampling interval. The number of bytes that arrive \( M(n) \) and the number of bytes that can be transmitted from the queue \( T(n) \) (i.e., available capacity), over the sampling period \( [(n-1)\Delta t, n\Delta t) \), \( n = 1, 2, ..., \) are measured. At discrete time \( n \), a new router's advertised window is computed by the following summation or integral control algorithm (see Fig. 2).

\[
W_{NE}(n+1) = W_{NE}(n) + \alpha (T(n) - M(n))
\]

\[
= W_{NE}(n) + \alpha e(n),
\]

where \( \alpha \) is a positive control constant and \( e(n) \) is the error (mismatch) between the target rate and the measured rate. The recursion (1) is a summation or integral control scheme since \( \Delta W_{NE}(n) = W_{NE}(n+1) - W_{NE}(n) = \alpha e(n) \) or \( W_{NE}(n) = \alpha \sum_{i=0}^{n} e(i) \), in discrete-time (and \( dW_{NE}(t)/dt = \alpha e(t) \) or \( W_{NE}(t) = \alpha \int_{0}^{t} e(\tau) d\tau \), in continuous-time). This simple algorithm lends itself naturally to recursive optimization of the router's advertised window.

The TCP Rate Control Scheme

When multiple TCP connections share a common link, the proposed approach matches the aggregate window sizes of all active TCP flows to the bandwidth-delay product of the network while at the same time providing all the connections with similar feedback to achieve fairness (in window allocation). The feedback is carried by returning TCP acknowledgments in the receiver's advertised window field. If the current value in the receiver's advertised window field, which is set by the destination system, exceeds the feedback value computed in the router, the receiver's advertised window is marked down to the feedback value. The computed feedback bounds the TCP window maintained at the source in order to limit packet losses in the router.

The difficult task here is the design of the feedback function to be implemented at the router. It should be noted that the dynamics of the system depends heavily on this function. The function can be designed to provide all TCP connections with similar feedback, and as a result have them operate with equal windows.

Our approach is to estimate the target (router's) advertised window size as a function of the input/output rate mismatch at the router buffer. The computed "router's advertised window" is then used to mark down the receiver's advertised window field in the acknowledgments. Since setting the window size too small can lead to starvation, a minimum window can be enforced.

Let \( W_{NE}(0) \) be an initialized router's advertised window for a given traffic class (in bytes). Fix \( \Delta t \) as the period of the sampling interval. The number of bytes that arrive \( M(n) \) and the number of bytes that can be transmitted from the queue \( T(n) \) (i.e., available capacity), over the sampling period \( [(n-1)\Delta t, n\Delta t) \), \( n = 1, 2, ..., \) are measured. At discrete time \( n \), a new router's advertised window is computed by the following summation or integral control algorithm (see Fig. 2).

\[
W_{NE}(n+1) = W_{NE}(n) + \alpha (T(n) - M(n))
\]

\[
= W_{NE}(n) + \alpha e(n),
\]

where \( \alpha \) is a positive control constant and \( e(n) \) is the error (mismatch) between the target rate and the measured rate. The recursion (1) is a summation or integral control scheme since \( \Delta W_{NE}(n) = W_{NE}(n+1) - W_{NE}(n) = \alpha e(n) \) or \( W_{NE}(n) = \alpha \sum_{i=0}^{n} e(i) \), in discrete-time (and \( dW_{NE}(t)/dt = \alpha e(t) \) or \( W_{NE}(t) = \alpha \int_{0}^{t} e(\tau) d\tau \), in continuous-time). This simple algorithm lends itself naturally to recursive optimization of the router's advertised window.
When a minimum window \( mW_{NE} \) is enforced at the router, then the recursion becomes

\[
W_{NE}(n+1) = \max\{mW_{NE}, (W_{NE}(n) + \alpha(T(n) - M(n))\}
\]  

Note that the receiver's advertised window can only be downgraded (if necessary) but should not be upgraded (since this may cause the data outstanding to exceed what a downstream node or a TCP receiver can handle). A maximum window should also be enforced. For instance, it can be set to a known maximum possible end-system advertised window such as the 16-bit window size TCP protocol field (65536 bytes).

TCP traffic can be very bursty, and in order to reduce the effects of inaccurate traffic measurements or effects due to transient changes in traffic pattern, various filters can be used to obtain smooth estimates. Moving average (MA) and exponential weighted moving average (EWMA) filters are two common examples. The MA filter can be used to obtain an estimate over the last \( L \) intervals as follows

\[
M(n) = \frac{1}{L}[\hat{M}(n) + \ldots + \hat{M}(n-L+1)] = \frac{1}{L} \sum_{k=n-L+1}^{n} \hat{M}(k)
\]

where \( \hat{M} \) denotes a measured value. The EWMA which exponentially weights old values may also be used. It is defined by the following recursion

\[
M(n) = \theta M(n-1) + (1-\theta)\hat{M}(n), \quad 0<\theta<1.
\]

In the proposed control scheme, the router's advertised window is used to modify the returning ACKs in a traffic class, regardless of the connections they belong to. That is, all connections in a traffic class (queue) are treated equally and receive the same feedback for the same network condition. This results in a simple control design and avoids the need to maintain the state of active TCP connections in the router. In the case of a connection not making use of its allocated window, there will be a mismatch (or error) between \( M(n) \) and \( T(n) \), causing an increase in the window signaled to all connections. This results in the active connections increasing their window sizes (thus their throughput), and sharing the available bandwidth.

SIMULATION RESULTS

The network topology used to evaluate the performance of the algorithm is shown in Fig. 3. The topology consists of two routers and several TCP source-destination pairs (i.e., connections).

Due to the limited space, only results for a bottleneck link capacity of T3 (45 Mbps) are presented here. All other non-bottlenecked links were given a capacity equal to the bottleneck link. In addition, all links have a propagation delay of 10 msec, resulting in each source having an RTT of 60 msec. All routers use a drop-tail packet discarding policy and their buffer sizes are set to twice the bandwidth-delay product (BDP) of the network (0.675 Mbytes). The sources implement the TCP-Reno version, which includes the Fast-Retransmit and Fast-Recovery mechanisms. The TCP connections are assumed to be greedy FTP connections; that is, they always have data to send. The maximum segment size (MSS) of TCP is set 1460 bytes. The TCP receiver's buffer size is set to the maximum 16-bit window size TCP protocol field. This value is also used as the upper bound for \( W_{NE}(n+1) \). The sssthresh variable of TCP is initialized to 65536 bytes. The OPNET Modeler was used for the simulation experiments.

The performance measures in the simulation studies are: normalized TCP goodput, router's queue size, and TCP segment number. The normalized goodput of each TCP source is computed as the number of good bits (after considering protocol overhead, link utilization factor, retransmissions, etc.) received in each 500 msec time interval, divided by the link capacity. For example, the maximum achievable normalized TCP goodput on a T3 link with packet size of 1500 bytes is 92.5% (based on a 95% link utilization factor). The router's queue size is measured after every 100-msec interval.

Results for A T3 Bottleneck Link Capacity

This section presents the simulation results for three scenarios when the bottleneck link is T3 (45 Mbps). The corresponding results for experiments not using the proposed scheme are also presented (i.e., TCP sources infer congestion solely by relying on packet losses and timeouts through which they reduce their transmission rates). The parameters of each scenario are given below. Since the number of TCP sources is large, and for resolution purposes we only plot the normalized goodput of Source 1 and every other fifth source thereafter (i.e., Sources 5, 10, 15, etc.).
TABLE I. SIMULATION SCENARIOS

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>TCP Source 1 to 10 start transmitting data at time 0.0 seconds</td>
</tr>
<tr>
<td></td>
<td>TCP Source 11 to 25 start transmitting data at 20.0 seconds and stop at 40.0 seconds</td>
</tr>
<tr>
<td></td>
<td>TCP segment size is set to 1460 bytes, target utilization = 95%</td>
</tr>
<tr>
<td></td>
<td>Δt = 100 msec, α = 0.025, initial window W_{neg}(0) = 5 packets</td>
</tr>
<tr>
<td>B</td>
<td>TCP Source 1 to 10 start transmitting data at time 0.0 seconds</td>
</tr>
<tr>
<td></td>
<td>TCP Source 11 to 25 start transmitting data at time 20.0 seconds</td>
</tr>
<tr>
<td></td>
<td>TCP Source 26 to 50 start transmitting data at time 40.0 seconds</td>
</tr>
<tr>
<td></td>
<td>All other parameters are as in Scenario A</td>
</tr>
<tr>
<td>C</td>
<td>TCP Source 11 to 25 start transmitting data at 20.0 seconds and stop at 40.0 seconds</td>
</tr>
<tr>
<td></td>
<td>TCP segment size is set to 1460 bytes, target utilization = 95%</td>
</tr>
<tr>
<td></td>
<td>Δt = 100 msec, α = 0.025, initial window W_{neg}(0) = 5 packets</td>
</tr>
</tbody>
</table>

Fig. 4, 7, and 10 show the normalized TCP goodput of Scenario A, B and C, respectively. These figures show how well the TCP rate control algorithm can share the available bandwidth when compared to the cases without the algorithm. The algorithm is responsive and fair in reallocating the bandwidth during the departure and entry of many connections. Fig. 10a shows how some of the connections in slow-start or congestion avoidance phases react to the feedback given by the algorithm. Although some connections are in congestion avoidance phase (due to packet loss), all connections eventually receive their fair share. This transient period depends on the rates at which the congestion windows of the TCP connections evolve. The figure also shows that the algorithm is effective at stabilizing the system independently of the connection's phase.

Fig. 5, 8, and 11 show the TCP segment number of Scenario A, B, and C, respectively. It can be seen from these figures that without the algorithm, there is no fairness in the allocation of the transmission rates (i.e., available bandwidth). In fact, the source rates fluctuate irregularly over time. Some connections then become "unlucky" (i.e., knocked down to smaller transmission rates) as shown in Fig. 8b.

Fig. 6, 9, and 12 show the queue size of Scenario A, B, and C, respectively. They indicate that the algorithm is effective at minimizing losses and queue fill. The transient overshoots in the queue size (which die out quickly with the algorithm) are due the entry of many connections all at the same time. However, in reality connection start-times (or stop-times) are most likely to be staggered.

CONCLUSIONS

We have described in this paper a TCP window control scheme which sends explicit feedback to TCP sources to adjust their window sizes. The scheme which can be implemented in a router allows feedback to be carried to the sources by returning TCP acknowledgments in the receiver's advertised window field. If the value of the receiver's advertised window, which is set by the destination system exceeds the feedback value computed by the router, the receiver's advertised window is marked down to the feedback value computed by the router. Using a simple recursive algorithm, we showed that the technique is able to adapt automatically to the number of active connections, the traffic load, the buffer occupancy, and the bandwidth-delay product of the network without maintaining any per-connection state. The scheme allows each active TCP source to achieve optimal performance by reducing traffic bursts, retransmissions, and lost packets.

We have investigated the behavior and characteristics of the algorithm through simulations. Results were shown for the typical bottleneck network configuration. In this paper, the algorithm was tested under large propagation delays, small and large amount of TCP sources and different link capacities. The results show that the algorithm can achieve fairness (in window size distribution), stability and robustness while providing low packet loss rates and minimal queue size. The algorithm was also tested (not show here due to lack of space) under different network topologies, small and large feedback delays, different link capacities (T1, T3 and OC3), and smaller amount of buffering at the bottlenecked router. In all cases, the algorithm achieved the desired results.

REFERENCES

Fig. 4. Scenario A goodput (T3 Link): (a) with TCP rate control algorithm (b) without.

Fig. 5. Scenario A TCP segment number (T3 Link): (a) with TCP rate control algorithm (b) without.

Fig. 6. Scenario A queue size (T3 Link): (a) with TCP rate control algorithm (b) without.
Fig. 7. Scenario B goodput (T3 Link): (a) with TCP rate control algorithm (b) without.

Fig. 8. Scenario B TCP segment number (T3 Link): (a) with TCP rate control algorithm (b) without.

Fig. 9. Scenario B queue size (T3 Link): (a) with TCP rate control algorithm (b) without.