Collaborative Congestion Control in Parallel TCP Flows

Soo hyun Cho and Riccardo Bettati
Computer Science Department
Texas A&M University
College Station, TX 77843 USA
Email: {s0c6496, bettati}@cs.tamu.edu

Abstract—We suggest a new congestion control scheme in parallel TCP flows and compare it with single-flow based congestion control approaches in the Internet. When a node opens multiple connections that are going to the same or nearby destinations, we can assume that there is a significant correlation of congestion events among the flows. In this paper we leverage this relationship to improve the performance provided to parallel TCP flows by having TCP senders of the related connections exchange congestion information. The usefulness of our scheme is shown by augmenting a measurement-based TCP congestion control mechanism to utilize congestion measurement information from other TCP flows at the sender. Furthermore, we compare our scheme with ECN-based TCP congestion control. Our measurement shows improvements over single-flow based approaches. In terms of packet loss and delay jitter, we show that collaborative congestion control is comparable with architecturally significantly more intensive schemes, such as ECN-based TCP.

I. INTRODUCTION

How to control or avoid congestion in the Internet has been an important topic of research for decades. For the stability and the efficient use of the Internet, it is necessary that every element of the Internet (end-hosts, servers and network equipments, etc.) operate collaboratively or should be forced to do so. The stability of the current Internet is mainly based on the end-to-end congestion control behavior [1] embedded in TCP. The de facto standard for the TCP congestion control is a loss-based approach, for example, in TCP-NewReno. By controlling the sender’s congestion window size in response to losses in the network, loss-based congestion control adjusts the amount of the outstanding data. However, any loss-based approach has the inevitable shortcoming that it cannot detect congestion until well after it happens.

To solve this problem many researchers proposed alternative methods to replace loss-based congestion control with delay-based schemes, such as Delay-based Congestion Avoidance (DCA) [2] or TCP-Vegas [3], or to combine delay-based fine tuning (such as TCP/Dual [4] and TCP/DCA [5]) with loss-based (so called, coarse) congestion control. TCP/Dual decides that congestion is impending if a round-trip time is larger than the average of maximum and minimum round-trip time. In TCP/DCA, sampled round-trip times are compared to long-term average and standard deviation of round-trip times to decide whether congestion is imminent. The rationale behind the various delay-based approaches is that the increase of round-trip time delay is due to queueing delay at switches and can be used as an indicator for impending packet losses. However, studies [5] show that the delay variation of a single TCP flow in the Internet is not sufficiently correlated with impending losses of the flow to be used for efficient congestion control.

Recent transmission technology improvements have resulted in much higher bandwidth available to Internet hosts thus significantly increasing the delay-bandwidth product over end-to-end connections. In this paper we use following observations. High-speed connections are often used for servers or very busy hosts, where multiple connections exist simultaneously. Similarly, in overlay networks, such as Content Distribution Networks (CDN), and Peer-to-Peer networks, it is common for much of the traffic to be forwarded between overlay nodes as flows aggregate. Flows in such systems tend to share long portions of paths (and bottlenecks), and benefits can be gained by sharing congestion information.

In order to do so, we group multiple TCP flows from a node according to their destinations, so that TCP flows in a group can share their congestion information which each other. We modify TCP flow control to adopt a simple delay-based congestion detection mechanism and make the flows exchange their congestion events within the group. By doing this, we expect that parallel TCP flows can (i) control their congestion windows (so, their sending rates) more accurately, so that they can (ii) reduce packet losses from flows in the group and (iii) maintain small and stable queue sizes in the network compared to those of unmodified parallel TCP flows.

The remainder of this paper is organized as follows: Section II surveys related work on parallel TCP flows. Section III presents the proposed collaborative congestion control scheme in parallel TCP flows with the definition of groups and events. Section IV shows that this scheme can be used to improve various performance measures of parallel TCP flows. Section V concludes this paper.

II. RELATED WORK

There has been significant research done to improve the congestion control when multiple flows are sending traffic to similar destinations from a single node. T/TCP (TCP for Transactions) [6] proposes temporal information sharing among parallel TCP flows. A T/TCP host caches previous TCPs’ congestion control information, e.g., Round Trip Time (RTT)
and Maximum Segment Size (MSS). The cached information is used when a new TCP connection is established to the same remote hosts. Ensemble-TCP [7] suggests both temporal and ensemble information sharing by using a common TCB (TCP Control Block) among concurrent connections. A new connection uses existing connection’s TCB and congestion control of all TCP flows are done based on an aggregated Congestion Window (cwnd).

Similarly, Congestion Management architecture (CM) [8] proposes an integrated congestion control and loss recovery scheme for parallel TCP connections. There is a single congestion window for the set of TCP connections between a sender and receiver. The integrated control block adjusts the total amount of unacknowledged data that the set of connections can put in the network. COCOON [9] has similar approach as ours in that it uses an aggregate control of a group of TCP flows. Specifically, it changes the congestion window size of a TCP flow whenever other flows in the group experience packet losses.

Our scheme is different from the above methods because each TCP flow in our scheme maintains its own congestion window and uses extra measurement information from other TCP flows in the group and responds to them independently. In addition, the particular realization described in this paper is based on a delay-based congestion avoidance mechanism as opposed to packet loss.

### III. Collaborative Congestion Control

In networks that do not provide congestion control mechanisms, such as Explicit Congestion Notification (ECN) [10], acknowledgement (ACK) packets are the only indicators for traffic senders to estimate the current network state. In an increasing number of service deployment scenarios, however, TCP flows get aggregated, and so share both large portions of their paths and likely the congested links on them. As a result, parallel flows would gain benefit by using other flows’ information in addition to their own to estimate the network status.

We use the term group to indicate a set of flows from a node that have the same destination node and that share the path between senders and destination nodes. This group concept can be extended to aggregated flows between clusters of nodes at the sender or the receiver side. For example, we can take advantage of IP’s hierarchical addressing [11] or systems like GNP [12] to cluster end-hosts, so that connections between clients share most of their path, thus allowing for an effective grouping of flows that not necessarily share both source and destination nodes. However, in this paper we only group flows with the same source and destination pairs.

Three types of congestion related indicators are available to the sending host: (a) round-trip time history, (b) duplicate acknowledgement (DUPACK) events, and (c) time-out events. While all three indicators can be used in collaborative congestion control, we focus on (a), round-trip time history. The rationale for this is that DUPACK and time-out events, while necessary for flow and error control for the affected flows, happen after congestion, which is typically too late for use for preventing packet losses of other TCP flows in the group. We therefore extend traditional TCP by adding an inter-flow, delay-based, congestion control method in addition to the existing loss-based congestion control mechanism.

We call our scheme TCP/DCA-C because (i) parallel TCP flows in our scheme adopt a delay-based congestion avoidance scheme and (ii) flows share delay events detected by TCP flows in the group as indicators of imminent congestion collaboratively (i.e., C stands for Collaborative). To detect imminent congestion, a TCP/DCA-C flow monitors its round-trip time, and use a threshold-based approach with the following threshold level $T$:

$$T = \text{rtt}_\text{min} + \gamma (\text{rtt}_\text{max} - \text{rtt}_\text{min}).$$

Here, $\text{rtt}_\text{min}$ and $\text{rtt}_\text{max}$ represent the maximum and minimum round-trip time of a TCP flow respectively. If a round-trip time is larger than the threshold, then a TCP flow in our scheme interprets this as an indicator for impending congestion. The value for parameter $\gamma$ used in this paper is $\frac{1}{2}$.

Fig. 1 illustrates the mechanics of TCP/DCA-C. As a basis, we use TCP-Reno. The TCP/DCA-C mechanism keeps track of RTTs to generate indicators of possible congestion. Flow grouping is done using a registration scheme: during connection establishment a new flow either creates and registers to the new group or register to an existing group. When a TCP flow is torn down it unregisters from the group. A TCP/DCA-C flow in a group behaves the same way as a normal loss-based TCP does, except that (i) it responds to delay-based congestion events as well if it is in congestion avoidance phase (i.e., $\text{cwnd} > \text{ssthresh}$), (ii) it passes its delay events to other flows in its group, and (iii) it reacts to incoming indicators from other flows. The value for the parameter $\alpha$, which decides the amount of reduction in congestion window of a TCP flow when it detects delay increase event from its own ACKs, is set to 0.125. This value is taken from the decrease ratio of congestion window suggested in DCA [2]. In DCA the sender decreases its congestion window if the normalized delay gradient is larger than 0.

When a TCP/DCA-C flow sends its delay event as a signal to other flows in the group, it also sends its current congestion window size. TCP/DCA-C flows in the same group reduce their congestion windows, but do so only if the congestion window of the signal sender is smaller than their own. Furthermore, TCP/DCA-C adjust the amount of congestion window reduction in response to indicator signals from other flows inversely proportional to the size of its group (i.e., the number of flows in the group). This compensates for group size effects, since more flows in the group generate more indicator signals.

It is reasonable to assume that expected sizes of congestion windows of all TCP/DCA-C flows in a group are similar because we define groups based on source-destination pairs and the path for a group is fixed. Furthermore, with a group of size $N$, we can reasonably assume that the number of flows sending congestion notifications with a smaller congestion window to a particular flow in the group is typically not more
At connection setup:
if (the group exists)
    register to it
else
    create a new group and register to it.

During connection:
same as TCP-Reno except for:
(1) when RTT updates
    update maxRTT and minRTT.
    if(RTT > minRTT + γ (maxRTT - minRTT)) |
        send a signal and cwnd to TCP flows in the group.
    if(cwnd > ssthresh)
        cwnd = cwnd - α * cwnd
    cwnd = cwnd - α * cwnd / (N-1)
(2) when a signal is received from other flows in the group of size N
    if(cwnd > signal sender’s cwnd)
        cwnd = cwnd - α * cwnd / (N-1)

At connection tear down: unregister from the group.

Fig. 1. Behavior of TCP/DCA-C

window sizes of TCP receivers were set to be large (e.g., 200 packets) enough for TCP senders to fill network resources.

FIFO Drop-Tail scheduling is used at the output of Node SW0 when we test TCP schemes that do not require network support. When we measure the performance of TCP/ECN flows, which require support from network, we configure an active queue management scheme at the bottleneck link. TCP/ECN responds to marked ACKs the same way it does to packet losses. For the ECN support in the network we run RED [15], one of the most popular active queue management (AQM) mechanisms, at the output buffer of Node SW0 and enable marking instead of dropping of packets. The parameters used for RED in all simulations are left to the default values of the ns-2 distribution except for a mean packet size of 1040 bytes. The default setting of RED gave us minth = 5, maxth = 15, w0 = 0.001663, and maxp = 0.1.

In the following, we compare the performance (in terms of goodput, packet loss rate, delay, and jitter) of TCP-Reno, TCP/DCA-S, TCP/DCA-C, and TCP/ECN for the case of no cross traffic, square-wave UDP cross traffic, and web cross traffic.

A. No Cross Traffic Case

In the first simulation we do not generate cross traffic in the test network shown in Fig. 2. We establish two FTP flows (Flow 0 and Flow 1) from the sender node S0 to illustrate the difference of the various schemes. In this simulation we compare the results of our collaborative congestion control scheme TCP/DCA-C and its variants (TCP/DCA-S) with unmodified TCP flows and TCP flows with ECN. A sender node S0 opens two FTP flows at the same time to a receiver node R0 for 50 seconds.

Fig. 3 shows the queue size changes in the bottleneck link from time 14 sec to 20 sec of the simulation run when we use different schemes for the two TCP flows. The graphs show that TCP/DCA-C maintains a small queue size and variation. With this result we expect that TCP/DCA-C will have low delay and jitter for packets arriving at the receiver. Unmodified TCP flows show large fluctuations of the queue size. The queue size of TCP/ECN is generally small as is to be expected, given the help from the network infrastructure.

In the figure we also show the congestion windows of the two parallel TCP flows in each scheme for the same time period. From the graphs we can see the TCP flows in TCP/DCA-C scheme maintain stable congestion windows. TCP/DCA-
S also shows stable congestion windows. However, we can see that there is significant difference in sizes of congestion windows between two TCP/DCA-S flows even though they are going to the same destination from the same source node. This shows that lock-out [16] is happening for TCP/DCA-S flows, which results in unfairness in throughput between the two flows. This lock-out occurs because TCP/DCA-S is a single-flow based scheme, where flows respond to their own congestion events only. This example illustrates the benefit of sharing congestion signals across flows.

In Table I we show performance measurement data of each scheme when two FTP flows start randomly from 0 to 0.1 sec and finish at 50 sec. All data shown in the table is obtained from averaging 10 simulation runs. The unit of aggregated goodput in the table is kbps and it is calculated by dividing the total number of packets arrived at the two FTP receivers by the simulation time. Except for TCP/ECN, the schemes do not show much difference in aggregated goodputs. The "Loss" column in the table shows percentage of lost packets of the two flows. Most of the losses of TCP/DCA-C flows happened during the initial slow-start phase. After they enter congestion avoidance phase they show little packet losses at all. Unmodified TCP and TCP/ECN showed significant packet losses during congestion avoidance phase as well.

The time unit in the "Delay" column is msec. The column shows average and standard deviation of one-way delays incurred by packets of the two TCP flows. In the column TCP/DCA-C flows show higher performance than TCP/ECN. To measure fairness in goodputs of the two flows we use Fairness Index from Chiu and Jain [17]. The measured data in the table show that single-flow based TCP/DCA-S is less fair than the other schemes, just as we expected from Fig. 3.

![Fig. 3. Queue Size and Congestion Window Changes](image1.png)

![Fig. 4. On-Off Cross Traffic, Different number of FTP Flows](image2.png)

**TABLE I**

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Goodput (avg./std.)</th>
<th>Loss (avg.)</th>
<th>Delay (avg./std.)</th>
<th>Fairness (avg.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>4705.29/6</td>
<td>0.41</td>
<td>71.8/19.6</td>
<td>0.997</td>
</tr>
<tr>
<td>TCP/ECN</td>
<td>4407.7/30.8</td>
<td>0.31</td>
<td>53.5/9.29</td>
<td>0.996</td>
</tr>
<tr>
<td>TCP/DCA-S</td>
<td>4784.8/7.9</td>
<td>0.29</td>
<td>68.0/5.41</td>
<td>0.972</td>
</tr>
<tr>
<td>TCP/DCA-C</td>
<td>4740.3/37.9</td>
<td>0.26</td>
<td>51.7/5.38</td>
<td>0.999</td>
</tr>
</tbody>
</table>

**B. On-Off CBR and Web Cross Traffic Cases**

In this section we first apply On-Off (i.e., square-wave) Constant Bit Rate (CBR) traffic over UDP as cross traffic in the setup described in Fig. 2. The CBR traffic comes from Node S1 and goes to Node R1 to disturb the TCP flows from Node S0 when it is On. The parallel TCP flows start randomly during the interval between 0 and 0.1 sec, and the On-Off CBR flow start in Off state at time 0 sec. The sending rate of the CBR flow when On is set to 60% of the bottleneck link, and durations of On and Off periods are both set to 10 sec. Intervals between CBR packets are randomly distributed to introduce jitters by enabling the random parameter for CBR in ns-2.

Fig. 4 shows the simulation results with various numbers of parallel TCP flows from Node S0 to Node R0. All the data shown in the figure are averages of 10 independent simulations of 50 second duration each. The figure shows that parallel TCP/DCA-C flows achieve high aggregate goodput and small packet loss ratios and jitters (we define jitter as standard deviation of one-way delays). One-way delays incurred by TCP/DCA-C packets are smaller than those of unmodified parallel TCP flows but larger than those of TCP/ECN flows. However, note that TCP/DCA-C scheme does not require support from the network nodes while TCP/ECN scheme does. In fact, TCP/DCA-C uses information acquired by TCP flows in its group at an sender-node only. In Fig. 5 we also show the results obtained when we changed bottleneck buffer sizes from 30 to 90 with 6 parallel TCP flows. The figure shows that the performance of TCP/DCA-C scheme is not affected by buffer size changes.
Fig. 6 shows the experiment results in the presence of web cross traffic instead of On-Off traffic. The cross traffic is established between web server nodes (S2 to S6) attached to the switch node SW0 and web client nodes (R2 to R6) attached to the switch node SW1. All the data shown in the figure are averages of 10 simulation runs obtained of 50 second duration each. For the simulation of web traffic we adopted a model suggested in [18]. In this model clients randomly initiate sessions to download files from server nodes. The parameters and distributions for the web traffic model used in simulations are: number of sessions: 100, inter-session-time: exponential with mean 5 sec, pages per session: 10, inter-page time: exponential with mean one sec, number of object per page: 10, inter-object time: exponential with mean 10 msee, and object size: ParetoII with mean 10 packets and shape 1.2. These parameters resulted in quite significant amount and variation of cross traffic in the test network. However, as we can see from Fig. 6, TCP/DCA-C shows similar performance to the previous On-Off cross traffic case in most aspects.

V. CONCLUSION

In this paper we proposed a new, measurement based, collaborative congestion control scheme called TCP/DCA-C for parallel, or quasi-parallel, TCP flows, which exchange indicator signals about imminent congestion within the group in order to improve performances of all the flows in the group. In TCP/DCA-C, flows in a group can manage their data sending rates more accurately to achieve better performance by taking advantage of information that comes from other TCP flows, which experience congestion earlier, and by treating their congestion signals as indicators of imminent congestion in network.

Simulation results show that TCP/DCA-C scheme for parallel TCP flows achieves better performance than unmodified parallel TCP flows in terms of packet loss ratio and delay and jitter compared to unmodified parallel TCP flows without compromising aggregated goodput. The scheme also showed improved fairness among parallel TCP flows and comparable performances to TCP/ECN scheme which requires support from network.

REFERENCES


Fig. 5. On-Off Cross Traffic, Different Buffer sizes

Fig. 6. Web Cross Traffic, Different number of FTP Flows