Aggregate Control of Parallel TCP Flows

Soo hyun Cho and Riccardo Bettati

Computer Science Department,
Texas A&M University,
College Station, TX 77843 USA
{so6496, bettati}@cs.tamu.edu

November 15, 2004

Abstract. The use of multiple concurrent parallel TCP flows is an easy way to achieve high-speed reliable data transfers. However, parallel TCP flows are inherently unfair with respect to a single TCP flow. We suggest a new scheme called TCP-P which controls aggressiveness of a group of parallel TCP flows by regulating their total aggressiveness (or unfairness) to be comparable to a single TCP flow, or any multiple thereof. TCP-P makes a group of \( N \) parallel TCP flows appear to other flows like \( k \) separable TCP flows - i.e., have strength \( k \) - through appropriate manipulation on increase and decrease behavior of the congestion windows of the TCP flows in the group. We implemented our scheme as part of Linux and experimental results show that the proposed scheme indeed effectively controls aggressiveness of parallel TCP flows.

1 Introduction

A widely used scheme to work around the limitations of TCP over high delay-bandwidth product connections is to use multiple parallel TCP connections. The use of parallel TCP flows has several benefits compared to a single TCP flow [1]. If an end-host opens \( N \) parallel TCP flows to the same destination, its congestion window recovery and increase are \( N \) times faster than a single TCP flow [2]. As a result, the achievable throughput of parallel TCP flows is significantly bigger than that of a single TCP flow with the same packet loss probability. Unfortunately, this increase comes at the expense of the throughput experienced by single TCP flows, as sender nodes who open multiple parallel TCP flows will consume unfairly more bandwidth when they compete for the same bottleneck links.

With the increased venues for bundling of TCP flows (e.g., overlay networks with TCP splicing [3], large servers with topological aggregation of service delivery, dedicated connections between supercomputers or campuses, etc.,) flexible schemes are needed for the controllable aggregation of large numbers of parallel TCP flows. Naively limiting the number of parallel connections applications in a
node can open concurrently is not appropriate in many situations, as it unduly violates the separation of application design from network resource allocation. Making the number of available connections visible to the application unduly burdens the application designs. On the other hand hiding the varying numbers of connection endpoints from the application through tunneling or multiplexing schemes typically is costly. Also, statically limiting the maximum aggregate sending rate of parallel TCP flows from sender nodes may leave network resources under-utilized because it disables TCP’s available bandwidth probing ability beyond the given sending rate.

Methods to control aggregation of TCP flows must have the following capabilities:

- Transparency to applications (management of connections and expected dynamics of data transmission should maintain TCP characteristics),
- Compatibility with existing TCP implementations (“TCP-friendliness”),
- Controllability and flexibility of the service (the “control knob” offered by the mechanism should be intuitive and have measurable effect on behavior).
- Effective use of available bandwidth (the mechanism should not prevent TCP from quickly making use of available bandwidth, in particular in high bandwidth-delay product situations),
- Flexible deployability (the mechanism should be deployable in single-sender (server), or multi-sender (overlay) scenarios).

In this paper we propose aggregate strength as means to control the fairness of parallel TCP flows: the aggressiveness or unfairness of parallel TCP flows appropriately controlled to not exceed that of a configurable number of single TCP flows, regardless of the number of TCP connections. By setting the aggregate strength of a group to some value $k$, the group of parallel TCP flows in a node behaves as if there were a group of $k$ parallel TCP flows regardless of the number of parallel flows applications in the node open. By doing this we improve the flexibility in aggregate control of parallel TCP flows while keeping the ability of parallel TCP flows to effectively utilize available bandwidth.

We realize the strength control as TCP-P, which is an extension to TCP. TCP-P controls the aggressiveness of a group of $N$ parallel TCP flows from a node against single TCP flows from other nodes by controlling the strength of the group of flows. The “strength” in this context is a scalar value $k$ of the TCP group and describes how big or aggressive the group is perceived by other TCP flows from other nodes sharing network resources with the group. We will show in the following how this parameter provides a simple and intuitive means to control aggressiveness of parallel TCP flows.

There have been several efforts to support parallel TCP flows to improve performance while constraining the unfairness of parallel TCP flows comparable to that of a single TCP flow. The Congestion Management (CM) architecture [4], Fractional/Combined TCP flows [5] and COCOON [6] are some examples. In contrast to these schemes, MultiTCP [7] was proposed to claim $k$ times more bandwidth for a single TCP connection. A MultiTCP flow with parameter $k$ increases and decreases its congestion window size as if there were $k$ multiple TCP
flows. However, as far as we know, there has been no scheme to controllably constrain the aggressiveness of parallel TCP flows.

The remainder of this paper is organized as follows: Section 2 presents the methodology we used to control parallel TCP flows' total strength. Section 3 describes our implementation in the Linux kernel. Section 4 presents experimental results that demonstrates how this implementation effectively controls the aggressiveness of parallel TCP flows. Section 5 presents throughput model of parallel TCP and TCP-P flows. Section 6 concludes this paper.

2 Aggregate Control

Aggregate control on parallel TCP flows is achieved through modifications to TCP's behavior while in the increase and the decrease phase. During the increase phase, a normal TCP has two modes: one is exponential increase mode during slow-start, the other is linear increase mode during congestion avoidance. Within the decrease phase, normal TCP responds to congestion events such as three duplicate acknowledgement packets (ACKs) by halving its congestion window size.

With a given strength parameter \( k \), we want to match the total amount of increase and decrease of congestion windows of a group of \( N \) parallel TCP-P flows to those of \( k \) single TCP flows. We denote the amount of increase of congestion window of a single TCP flow \( i \) in increase phase to be \( \Delta_{i}^{+} \) and the amount of decrease in decrease phase to be \( \Delta_{i}^{-} \), respectively. Let the amount of increase and decrease of a TCP-P flow \( j \) in a group of size \( N \) be \( \Delta_{j}^{P+} \) and \( \Delta_{j}^{P-} \) respectively. To make \( N \) parallel TCP-P flows become like \( k \) single TCP flows we need to make sure that \( \sum_{i=1}^{k} \Delta_{i}^{+} = \sum_{j=1}^{N} \Delta_{j}^{P+} \) for the given number of non-duplicate ACKs, and \( \sum_{i=1}^{k} \Delta_{i}^{-} = \sum_{j=1}^{N} \Delta_{j}^{P-} \) for a congestion event.

2.1 Controlling Increase

In slow-start mode, TCP increases its congestion window by one per non-duplicate ACK until it detects congestion events or the congestion window size reaches its slow-start threshold value. When a TCP is in slow-start mode, the congestion window size of a TCP, \( W \), after a non-duplicate ACK arriving at time \( t \) is shown in the following equation:

\[
W(t+) = W(t) + 1. \tag{1}
\]

When all TCP flows in a group of \( N \) unmodified parallel TCP flows are in slow-start mode, the total congestion window increase will be \( N \) times faster that a single TCP flow. For the same group size of TCP-P flows to have strength \( k \), the congestion window of each TCP-P flow, \( W_{j} \), should increase by \( \frac{k}{N} \) per non-duplicate ACK as shown in the following equation:

\[
W_{j}(t+) = W_{j}(t) + \frac{k}{N}. \tag{2}
\]
In congestion avoidance mode, we want to make the aggregate congestion window size increase of $N$ parallel TCP-P flows with strength $k$ be equal to that of $k$ TCP flows for the same amount of non-duplicate ACKs. We describe the special case of $k = 1$ first, and generalize it later. Let $W(t)$ be the congestion window size of a single TCP at time $t$, and we assume the sum of congestion window size of each TCP-P flow in the group is equal to $W(t)$, i.e., \( \sum_{j=1}^{N} W_j(t) = W(t) \). The amount of congestion window increase of a single TCP per non-duplicate ACK in this mode is $\frac{1}{W(t)}$ as shown in the following equation:

\[
W(t+) = W(t) + \frac{1}{W(t)}.
\]  

(3)

If each TCP-P flow in a group size $N$ increases its congestion window by one, the total increase will be $N$. For a single TCP flow to increase its congestion window size $W$ by $N$, it needs $W + (W + 1) + (W + 2) + \cdots + (W + N - 1) = \sum_{i=0}^{N-1} (W + i)$ non-duplicate ACKs. Hence, to match the congestion window increase speed of $N$ parallel TCP-P flows to that of a single TCP flow, we should require the group of TCP-P flows with size $N$ to receive $\sum_{i=0}^{N-1} (W + i)$ non-duplicate ACKs before each TCP-P flow in the group increases its congestion window by one.

To ensure fairness among TCP-P flows within the group, we evenly distribute the total amount of non-duplicate ACKs required for a group to each TCP-P flow in the group, so that each TCP-P flow in a group needs to receive $\frac{\sum_{i=0}^{N-1} (W + i)}{N}$ non-duplicate ACKs before it can increase its window size by one. In this way, with the same amount of non-duplicate ACKs, i.e., $\sum_{i=0}^{N-1} (W + i)$, the $N$ parallel TCP-P flows will increase their total congestion window size by the same amount, $N$, just as the single TCP flow.

For the case of $k > 1$, we generalize the previous case: parallel TCP-P flows need to increase their total window size of the group by $k$ after the group received $\sum_{i=0}^{N-1} (kW + i)$ non-duplicate ACKs. Here, $W$ is the average of the congestion window sizes of $k$ TCP flows, and we assume that $\sum_{i=1}^{k} W_i = kW = \sum_{j=1}^{N} W_j$, i.e., the sum of congestion window $W_i$ of $k$ single TCP flows is equal to the sum of congestion window $W_j$ of $N$ parallel TCP-P flows at time $t$. We use the fact that the increase of the total congestion window size of $k$ parallel TCP flows with a given number of non-duplicate ACKs is $k$ times larger than the increase of the congestion window of a single TCP flow with the same window size (i.e., $kW$). For this, each TCP-P flow in a parallel TCP-P group of size $N$ should increase its congestion window by one after receiving the following amount of non-duplicate ACKs:

\[
\frac{\sum_{i=0}^{N-1} (\sum_{j=1}^{N} W_j + i)}{k \cdot N},
\]  

(4)

As a result, the increase behavior of a group of $N$ TCP flows can be made to closely reflect that of $k$ TCP flows.
2.2 Controlling Decrease

A single TCP flow reduces its congestion window size $W$ by half when it detects a congestion event such as three duplicate ACKs at time $t$:

$$W(t+)=\frac{W(t)}{2}. \quad (5)$$

In unmodified parallel TCP flows, only one TCP flow will halve the congestion window size for each congestion event to the group. This behavior primarily contributes to the observed throughput advantage (and the unfairness) of parallel TCP flows over a single flow. In contrast, we let each TCP-P flow in a group respond to its own congestion event the same way as a normal TCP flow does. In addition, TCP-P adjusts congestion windows of other TCP-P flows in the group as well based on the group size $N$ and strength parameter $k$.

For $k=1$, we must halve all parallel TCP-P flows’ congestion window sizes whenever any member flow detects a congestion event. For $k>1$, we let the total congestion window size after a congestion event be $\frac{2k-1}{2k}$ of the previous total congestion window size of $N$ parallel TCP-P flows. Hence, for strength $k$, TCP-P scheme controls the total amount of congestion window decrease of a group of parallel TCP flows according to the following equation:

$$\sum_{j=1}^{N} W_j(t+) = \sum_{j=1}^{N} W_j(t) \times \left(\frac{2k-1}{2k}\right). \quad (6)$$

For $k=N$, $N$ parallel TCP-P flows becomes unmodified $N$ parallel TCP, and the total decrease amount of $N$ TCP-P flows become $\frac{2N-1}{2N}$ of the previous total window size that is the same as that of unmodified parallel TCP flows' [2].

2.3 Avoiding Unnecessary Decreases

Since packet drops in the network are typically bursty [8], multiple TCP flows in a parallel TCP group may experience packet losses at the same time. If bursty packet drops occur, they may result in too much congestion window reduction to parallel TCP-P flows: every TCP-P flow that experiences a congestion event might in turn trigger a congestion window reduction in other flows, which already may have responded to the congestion event. This results in a congestion response cascade. In traditional a TCP flow responds to congestion events only once within a congestion window, regardless of the number of lost packets. In comparison, TCP-P flows may end up with a lower throughput than that of a single TCP flow.

To avoid this unnecessary reduction of congestion windows, TCP-P does not adjust congestion windows of other member TCP-P flows if the elapsed time after its last adjustment is less than half of its average round-trip time. By doing this we assumed the effect of bursty drops will cease in half of the average round-trip time.
3 Implementation Issues

We implemented TCP-P scheme on Redhat Linux 9.0 kernel 2.4.20-8. The default behavior of Linux TCP implementation is based on TCP-Sack [9], time-stamping on each packet, and delayed ACK [10] in congestion avoidance mode (called Quick-ACK) [11]. Also, Linux uses the packet as the unit of congestion window size unlike BSD, which uses bytes.

3.1 Structure

Whenever a TCP connection is established, the system kernel looks up the group list using the destination IP address as a key to know whether there are already other flows exist to the same destination\(^1\). If no such group exists, a new group entry is added in the list and the connection is registered as a member of the group using its sock structure pointer. Otherwise, the connection is added as a member to the group. When a connection closes, the connection is removed from the list of members of the group. If the group has no more members it is also deleted.

Fig. 1 illustrates how the Linux data structures are extended to manage TCP flow groups. The Linux TCP implementation has a structure named sock to manage socket information for each connection and tcp_opt for TCP specific information. We added new variables to those structures for TCP-P scheme: a pointer that points flow_num variable of its group is added to tcp_opt to get the number of flows of its group without searching the group list, and a pointer to the next sock structure in the same group is added in the sock structure.

\(^1\) In this implementation, we use destination address as group classifier. Other classification could be used just as well.
Each group structure has a pointer to its first member’s structure sock and has a integer variable flow_num to count the number of member flows of the group. Whenever a new member TCP is added or deleted in a group this count variable updates the number of member TCP flows. For a system-wide control of strength we added a new system control parameter syscall.tcp_strength in net/ipv4/sysctl_net_ipv4.c. This parameter can be easily changed in the run-time using the syscall system call.

3.2 Implementing Increase

In slow-start mode, the congestion window of each TCP-P flow in a group is increased by $\frac{k}{N}$ per non-duplicate ACK as Equation (2). The amount of increase, $\frac{k}{N}$, is less than or equal to 1 when $k$ is not bigger than $N$. Since floating point arithmetic is not supported in the Linux kernel we let each TCP-P flow in our scheme increase its congestion window size by $k$ after receiving $N$ non-duplicate ACKs.

In congestion avoidance mode, each TCP-P flow in a group of size $N$ with strength parameter $k$ should increase its congestion window by 1 after receiving $\sum_{j=1}^{N} \sum_{i=0}^{N-1} W_j + \sum_{i=0}^{N-1} i$ non-duplicate ACKs as Equation (4). To implement this for each TCP-P flow independently, we use the following equation:

$$\frac{\sum_{j=0}^{N-1} \sum_{j=1}^{N} W_j + \sum_{i=0}^{N-1} i}{k * N} = \frac{N \sum_{j=1}^{N} W_j + \sum_{i=0}^{N-1} i}{k * N}.$$  \hspace{1cm} (7)

Therefore, each TCP-P flow should increase its congestion window size $W_j$ by 1 after $\frac{N \sum_{j=1}^{N} W_j + \sum_{i=0}^{N-1} i}{k * N}$ non-duplicate ACKs or it can increase the congestion window by $\frac{N \sum_{j=1}^{N} W_j + \sum_{i=0}^{N-1} i}{k * N}$ per non-duplicate ACK.

Looking up other TCP-P flows’ congestion window size at every non-duplicate ACK arrival may result in serious overhead. To reduce operation cost we use an assumption that all TCP-P flows in a group have the same window size $W_0$, so that $\sum_{j=1}^{N} W_j = NW_0$. With this assumption, each flow does not need to know other TCP-P flows’ congestion window sizes. Instead, it can use its own congestion window $W_j$ to estimate total window size for the group. Each TCP-P can find the size of its group, $N$, easily because each TCP-P structure has a pointer to its group’s member count variable flow_num as shown in Fig. 1.

Since floating point arithmetic is not supported in Linux kernel, we increase the congestion window size of each flow by $k$ after it received $\sum_{j=1}^{N} W_j + \sum_{i=0}^{N-1} i$. 
non-duplicate ACKs. This number can be further simplified as follows:

\[
\frac{N^2 \cdot W_j + \sum_{i=0}^{N-1} i}{N} = NW_j + \frac{(N - 1)N}{2N} = NW_j + \frac{N - 1}{2}.
\]  

(8)

Therefore, each TCP-P flow in a group of size \( N \) and strength \( k \) should increase its congestion window by \( k \) after receiving \( NW_j + \frac{N-1}{2} \) non-duplicate ACKs.

3.3 Implementing Decrease

TCP-P scheme controls the decrease amount of total congestion window sizes of parallel TCP-P flows according to Equation (6) to match with that of \( k \) unmodified parallel TCP flows. One possible method to implement this is to decrease every TCP flow’s congestion window by the same proportion. However, in TCP-P scheme, whenever a TCP-P flow \( i \) detects a congestion event at time \( t \), it responds like a normal TCP: It enters recovery mode and halves its own congestion window regardless of other parallel flows. Therefore, other TCP-P flows in the group can reduce their congestion window sizes less than the proportion shown in Equation (6) when strength \( k > 1 \).

Hence, the amount of decreases of congestion window sizes of other member TCP-P flows' become as follows:

\[
W_j(t+) = W_j(t) \cdot \left( \frac{1}{2} + \frac{N(k - 1)}{2(N - 1)k} \right), \forall j \neq i.
\]  

(9)
This equation is derived from the following equation to distribute the remaining amount of congestion window decrease to other member TCP-P flows.

\[
\sum_{j=1}^{N} W_j \left( \frac{2k - 1}{2k} \right) = W_0 N \left( \frac{1}{2} + \frac{k - 1}{2k} \right) = W_0 (1 + N - 1) \left( \frac{1}{2} + \frac{k - 1}{2k} \right) = \frac{1}{2} W_0 + W_0 (N - 1) \left( \frac{1}{2} + \frac{N(k - 1)}{2(N - 1)k} \right).
\] (10)

4 Evaluation

For the evaluation of TCP-P scheme, we used the topology shown in Fig. 2. To emulate delays and packet losses in the Internet, we use NIST Net Emulator [12]. The NIST Net Emulator is implemented on a Linux machine and emulates the Internet by appropriately delaying and dropping packets. Because the network links we experiment with are fairly high-bandwidth (default 100Mbps) and the NIST Net delay parameters are large (50msec round-trip propagation delay) we set the TCP parameters of the Linux end systems - such as tcp_rmem and tcp_wmem sizes - appropriately rather than using system defaults. We also disabled the TCP time-stamping and TCP-Sack options to see the effects of our modification more clearly. By disabling TCP-Sack, Linux TCP works based on TCP-NewReno.

All the end-host nodes and the NIST Net Emulator are running on Linux PCs. The PCs we use for experiments are Pentium 4 or 3 machines with 10/100 Mbps Fast Ethernet network interface cards. Each Fast Ethernet card has an output queue of length 100 packets by default, which can be controlled if needed. Two TCP sender nodes, Node 0 and Node 1, run both on Redhat 9.0 with kernel 2.4.20-8. In machine Node 0 we installed a modified Linux kernel that supports TCP-P scheme when we set \( k > 0 \). NIST Net Emulator and the TCP sink, Node 2, are running on Redhat Linux 7.2 with kernel 2.4.7-10.

For traffic generation and throughput measurement we use iperf, which supports parallel TCP flows and offers great flexibility for measurements. In all experiments in this section, every experiment was done for 100 sec to get an average value and repeated 10 times with 5 sec waiting time after each experiment. Error bars at figures of this section represent 95% Confidence Interval of the data.

4.1 \( k = 1 \)

We first show the performance of TCP-P with \( k = 1 \). We open a group of parallel TCP-P flows from modified Linux kernel at Node 0 to a TCP sink Node 2 for
100 seconds, and a single unmodified TCP flow from another sender Node 1 to Node 2 for the same time. Fig. 3 (a) shows the experimental results with a varying number of parallel TCP-P flows from Node 0 with $k = 1$. This figure shows that the average of aggregated throughput of a group of parallel TCP-P flows of $k = 1$ with group size from 1 to 10 remains comparable to the average throughput of the single TCP flow from Node 1.

In order to investigate the robustness of the TCP-P approach we repeated the experiments with a reduced bottleneck link speed by limiting the link speed from NIST Net Emulator to TCP sink node Node 2 from 100Mbps to 10Mbps. Fig. 3 (b) shows the experiment results with 10Mbps link. These results also show that TCP-P can regulate the aggressiveness of parallel TCP-P flows not to steal bandwidth from the single TCP flow, so that the throughput of the single TCP flow from Node 1 is comparable to that of total parallel TCP-P flows from Node 0 regardless of the group size $N$.

Figures in Fig. 4 illustrates some of the details of the operation of TCP-P. These figures are generated by tcptrace using tcpdump data on the sender Node 0 and Node 1. For the simplicity of comparison of the time-dependent behaviors we open two parallel TCP-P flows with $k = 1$, Flow 0 and Flow 1, from Node 0 to Node 2, and one TCP flow, Flow 2, from Node 1 to Node 2. Other conditions are the same to the previous experiment, and all connections start at the same time and finish after 100 seconds. Average throughput achieved by two TCP-P flows from Node 0 and a single TCP flow from Node 1 were 4.73Mbps and 4.78Mbps respectively.

The figures show the amount of outstanding data of each TCP flow, from which we can infer the changes of congestion windows of TCP flows. The spikes
in the figures represent Fast-Retransmission behaviors of TCP flows. Fig. 4 (a) and Fig. 4 (b) are for two TCP-P flows from Node 0. We can see in these figures that there are congestion window decreases without spikes, which indicates adjustments of the congestion window by the other TCP-P in the group. Compared to these two figures, the change in the congestion window of Flow 2 in Fig. 4 (c) always have a spike before a reduction.

4.2 \( k > 1 \)

In the following, we illustrate the aggregate control by TCP-P scheme actually controls the magnitude of aggressiveness of parallel TCP flows according to the strength parameter \( k \). We first present experiment results with unmodified parallel TCP flows in Fig. 5 (a). Node 0 opens \( N \) unmodified parallel TCP flows to Node 2 for 100 seconds, while Node 1 opens a single TCP flow to Node 2 for the same time. Fig. 5 (a) shows the average throughput of TCP flows from Node 0 and Node 1 for a varying number of unmodified TCP flows from Node 0. The single TCP flow from Node 1 achieves increasingly smaller throughput with increasing numbers of unmodified parallel TCP flows from Node 0. It illustrates the unfairness of parallel TCP flows mentioned in Sec. 1.

In comparison, Fig. 5 (b) shows the results of TCP-P at the same environment except that we let Node 0 open a group of 10 parallel TCP-P flows to Node 2 with varying strength \( k \). Fig. 5 (b) shows average throughput of TCP-P flows and a single TCP flow when we control the aggressiveness of the group of parallel TCP-P flows. In the figure, with \( k = 0 \) we describe the case of no parallel TCP-P flows sending any traffic to the destination, so that only the single flow from Node 1 consumes all bandwidth. By comparing (a) and (b) in Fig. 5 we see that TCP-P scheme accurately controls the overall aggressiveness of a group of 10 parallel TCP-P flows according to \( k \). 10 TCP-P flows with strength \( k \) show almost the same effect to a single TCP as \( k \) unmodified parallel TCP flows.

Fig. 5 (c) shows the ratios of total achieved throughput of parallel TCP flows against a single TCP flow. The solid line in the figure presents the ratio between
aggregate throughput of 10 TCP-P flows with different strength $k$ and that of a single TCP flow. The dotted line in the figure shows the ratio between the aggregate throughput of $N$ unmodified parallel TCP flows and that of a single TCP flow. Both graphs show the ratios closely follow Equation (14) shown in Sec. 5.1.

5 Throughput Modeling of Parallel TCP and TCP-P flows

In this section, we propose a steady-state throughput model of $N$ unmodified parallel TCP flows and $N$ parallel TCP-P flows with strength $k$. We compare the proposed model to that of a single TCP flow, which has been extensively studied (e.g., [13], [14], etc.). Previously the throughput of $N$ parallel TCP flows was empirically known as proportional to the number of parallel flows, i.e., $N$ times larger than a single TCP flow [5]. We evaluate the proposed throughput model of parallel TCP flows by measuring the ratio of consumed bandwidth between parallel TCP flows and a single TCP flow with the testbed in Sec. 4.

A similar result to our modeling has been reported in the modeling of throughput for MuTCP [7] because its assumptions were almost same as ours. However,
it was for the throughput modeling of MulTCP not for the parallel TCP or TCP-P flows.

5.1 Modeling

Because a TCP flow in unmodified parallel TCP flows behaves the same way as a normal TCP flow does, in congestion avoidance mode it reduces its congestion window size by half if there is a packet loss, otherwise it increases its congestion window by one per round-trip time. To model the aggregate throughput of $N$ parallel TCP flows we closely follows the method used in [13]. We first assume that all TCP flow in a group has the same congestion window size $W$ at time $t$, so that the sum of all congestion windows of the parallel TCP flows is then $NW$. We also assume that the packet loss probability is very small, so that only one packet loss happens per congestion epoch (we call a period that starts from window size with $NW$ and ends at the next $NW$ as a congestion epoch), and delayed acknowledgement is not used, i.e., one ACK per one data packet is generated by the receiver.

The total increase amount of congestion window of $N$ parallel TCP flows is $N$ per round-trip time instead of one of a single TCP flow if there is no packet loss. Also, the reduction of total congestion window is only $1/2N$ of total congestion window ($NW$) instead of halving of the congestion window of a single TCP [2]. Hence, in parallel TCP flows the time needed to recover the congestion window (period of the congestion epoch) is $\frac{W}{2N}$ round-trip times because the amount of windows size needed to recover is only $\frac{W}{2}$ as shown in Fig. 6.

Let $S$ be the total amount of packets sent in each congestion epoch. From Fig. 6 we can see that $S = NW(1 - \frac{1}{2N}) + \frac{W}{2N}$. If we let $p$ be the packet loss probability then, by the assumption, $p$ equals to $\frac{1}{S}$. Hence, $\frac{1}{p}$ can be expressed
using the following equation.

\[
\frac{1}{p} = S
\]

\[
= \frac{W}{2N} (NW \left( \frac{2N-1}{2N} + \frac{W}{4} \right))
\]

\[
= \frac{W^2}{2N} \left( \frac{2N-1}{2} + \frac{1}{4} \right)
\]

\[
= \frac{W(4N-1)}{8N}.
\] (11)

As a result, \( W = \left( \frac{1}{p} \cdot \frac{2N}{4N-1} \right)^{1/2} \). Therefore, the average aggregate throughput \( T(N) \) [packets/sec] of \( N \) parallel TCP flows can be modeled as follows:

\[
T(N) = \frac{\text{data sent per epoch}}{\text{time per epoch}}
\]

\[
= \frac{W^2(4N-1)}{8N} \cdot \frac{N}{RTT}
\]

\[
= \frac{W(4N-1)}{4RTT}
\]

\[
= \left( \frac{4N-1}{8N} \right)^{1/2} \cdot (4N-1)
\]

\[
= \frac{\sqrt{N(4N-1)}}{2} \cdot \frac{1}{\sqrt{p}}
\]

\[
= \frac{\sqrt{N(4N-1)}}{2} \cdot \frac{1}{\sqrt{p}} \text{ [packets per sec].}
\] (12)

This equation can also be found by setting the AIMD parameter to be \( a = N \) and \( b = \frac{1}{2N} \) in the AIMD(a,b) equation of \( T = \frac{\sqrt{ab}}{\sqrt{b} \cdot b \cdot \sqrt{a}} \) given in [14]. We can also compare the result when \( N = 1 \) with the following throughput equation of a single TCP flow given in [13].

\[
T(1) = \frac{\sqrt{3/2}}{RTT \cdot \sqrt{p}} \text{ [packets per sec].}
\] (13)

Equation (12) shows that with the same round-trip time and packet loss probability, \( N \) parallel TCP flows can achieve \( \sqrt{N(4N-1)/3} \) more throughput than a single TCP flow.

\[
\frac{T(N)}{T(1)} = \sqrt{N(4N-1)/3}.
\] (14)

We will evaluate this ratio model in Section 5.2 by comparing average throughput of \( N \) parallel TCP flows with that of a single TCP flow.

In contrast to the unmodified TCP flows, \( N \) parallel TCP-P flows with strength \( k = 1 \) controls the total congestion window increase and decrease of parallel TCP-P flows to match that of a single TCP flow, so that it results in
effective AIMD parameters for parallel TCP-P flows with $a = 1$ and $b = 1/2$, which are equal to those of a single TCP flow. Therefore, The throughput model of TCP-P with $k = 1$ is given by Equation (12) with $N = 1$. Also, with strength $k > 1$, from the behavior of TCP-P described by Equation (4) and (6), it is obvious that the throughput model of $N$ parallel TCP-P flows with strength $k$ is given by Equation (12) with $N$ replaced by $k$.

### 5.2 Evaluation

We configure the test bed shown in Fig. 2 to have all links be 100Mbps. Despite the plentifully available bandwidth, a single TCP flow alone achieves about 75Mbps in average. This achievable bandwidth is significantly reduced when NIST Net invoked errors cause random packet losses. When we configure NIST Net to drop 0.01% of packets, the achievable bandwidth for a single TCP flow drops to 20Mbps. This value is close to the theoretical average throughput of a TCP flow [13] if we consider delayed-ACK of Linux TCP and a maximum segment size of 1500 bytes.

Again, Node 0 opens a varying number of TCP flows to Node 2 while Node 1 opens only a single TCP flow to Node 2. We first show the experiment results for unmodified TCP flows in Fig. 7 (a). This figure shows the average throughput of unmodified parallel TCP flows that share the link to Node N2. We see from these results that increasing the number of unmodified parallel TCP flows from Node 0 gives rise to high throughput even if there is a high percentage of non-congestion packet losses. However, this high throughput is achieved by *stealing* the bandwidth from the single TCP flow from Node 1 starting from four parallel TCP flows and more.
Fig. 7 (b) shows the ratio of the total achieved throughput of unmodified parallel TCP flows from Node 0 to the throughput of the single TCP flow from Node 1. This figure shows the throughput ratio of experiment results are indeed similar to the ratio model given in Equation (14).

6 Conclusion

In this paper, we proposed TCP-P for aggregate control on parallel TCP flows, which extends TCP with controllable effect on unfairness to flows from other nodes. TCP-P scheme uses strength as a single - easily tunable - parameter to accurately control the aggressiveness of a group of TCP flows with respect to a single flow sharing the same bottleneck link. We showed that by employing TCP-P we can control the total aggressiveness or unfairness of parallel TCP flows against TCP flows from other nodes in a easily parameterizable and controllable way without requiring application modification. For future work, we are considering an adaptive control of strength of parallel TCP flows adequate for the efficient use of network while satisfying various requirements of applications.

References