# The Case for Additive Increase in Slow Start

Lachlan L. H. Andrew

***Abstract*— In cases where a TCP flow knows approximately what window it will have when slow start finishes, additive increase is less disruptive to other flows and reduces the transfer time of short bursts.**

This document is a work-in-progress.

## I. Introduction

When a TCP connection starts, it transmits slowly by using a congestion window of around 1 to 4 packets, which is doubled each round trip time (RTT) until congestion is encountered. This process is called slow start. The process of increasing the window by a constant factor each RTT is called multiplicative increase (MI); specifically, let MI-2 refer to doubling the window each RTT.

Note that the bitrate achieved by a flow is proportional to its window size; this paper focuses on changes in window size rather than bitrate to avoid confusion between the bitrate and rate of *increase* of window or bitrate. Time will be measured in multiples of the RTT.

This MI is suitable if the sender has no idea of what window to target, because the "rise time" only increases logarithmically with the final window size and the maximum rate of increase in the window is also proportional to the final window size, whatever the capacity of the network is.

However, exponential increase is very fast when the fair window is reached, causing interruption to other flows, while being very slow initially, causing unnecessarily long start up for short flows.

Vegas [1] was originally proposed to use additive increase for slow start to avoid packet loss. Since a new connection does not know the final window size, it used a very conservative rate of additive increase, causing connections to start very slowly. The Vegas implementation in Linux uses regular exponential slow start.

However, in several cases, the sender already has some information about the window size at which slow start should stop. For example,

- explicit signalling schemes like MaxNet [2], [3], RCP [4] and Quick Start [5]
- after an idle period in TCP friendly rate control (TFRC) [6].

It is recognised in[7] that slow start is unnecessarily conservative in those cases, and a faster exponential increase has been proposed, until the estimated capacity is reached.

These cases allow exponential increase to be replaced by an increase which is both less disruptive for competing flows and also faster for short transfers.

| | MI-2 | AI |
|---|---|---|
| Maximum rate of change | $W/2$ | $(W-1)/\log_2(W)$ |
| Time to send $W/4$ | $\log_2(W)-2$ | $\log_2(W)/2$ |

TABLE I

COMPARISON OF TRADITIONAL DOUBLING AND ADDITIVE INCREASE FOR SLOW START.

## II. Conservative additive increase

The harm caused to other flows when a flow increases its window is approximately proportional to the absolute rate of increase. In practice, it may even depend on the fractional rate of decrease of remaining capacity; for a given absolute rate of increase, this quantity increases faster for larger windows. These are both in contrast to the assumption that the harm is proportional to the *relative* rate of increase of window, as implicitly assumed by MI; for a given absolute rate of increase, this quantity increases slower for larger windows.

This section addresses the optimisation problem: Minimise the maximum rate of change of window size during slow start subject to the constraint that the time to reach the final window is given. The converse optimisation of minimising the time to reach the final window given a constraint on hte maximum rate of change of window size is addressed in Section III.

The above optimisation is solved by increasing the window at a constant rate throughout slow start. The duration of slow start using MI-2 starting from a window of 1 is $\log_2(W)$. The optimal solution is thus to use additive increase (AI), with the window increasing by $(W-1)/\log_2(W)$ each RTT. Call this approach AI-2.

As well as minimising the maximum rate of change of window, this AI reduces the time taken to transmit a burst which finishes before slow start is complete. This may be either a short flow in the context of explicit signalling [2], [3], [4], [5], or a short burst after an idle period in TFRC.

To illustrate this, Table I shows both the maximum rate of increase in window, and the time taken for a burst of $W/4$ packets to be transmitted. These metrics are improved by factors of approximately $\log_2(W)/2$ and 2, respectively. In general, a flow of length $W/k$ is improved by a factor of $\sqrt{k}$.

This is illstrated in Figure 1

## III. More aggressive options

This work was motivated by the proposal [7] to quadruple the window every RTT after an idle period in TRFC, up until a certain threshold. This results in halving the total time. Call this algorithm MI-4, and let AI-4 be the additive increase algorithm which finishes at the same time, after $\log_4(W)$ RTTs.

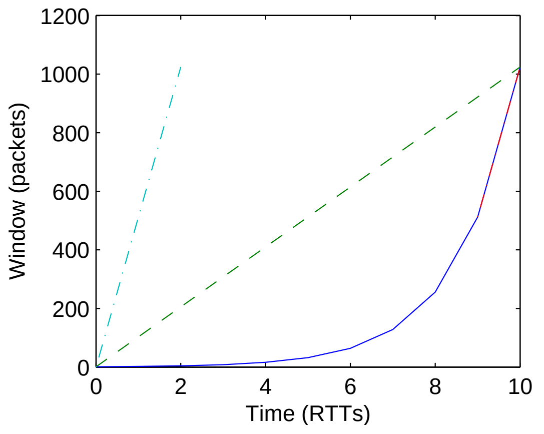

Fig. 1. Window size as a function of time, for a slow start up to $W = 1024$ packets. Solid line shows traditional MI-2, dashed line shows proposed AI, and dot-dashed line demonstrates the maximum rate of increase arising from MI-2.

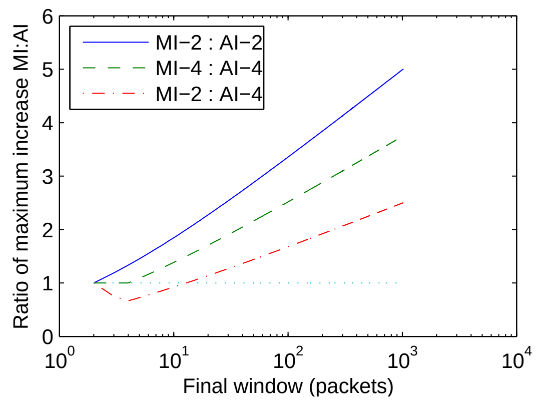

Fig. 2. Ratio of maximum rate of window increase using MI to maximum rate using AI

Figure 2 shows how much more disruptive the MI schemes are than AI, measured as the ratio of the maximum change in window over a RTT during slow start. The first two curves compare AI and MI schemes which finish at the same time; in this case, MI always has a higher peak rate of increase. For $W \leq 4$, both MI-4 and AI-4 increase to $W$ directly after one RTT, causing the flat start to the curve for MI-4:AI-4. The final curve compares AI-4 with the traditional MI-2; even though AI-4 finishes twice as soon as MI-2, it has a lower peak rate of increase for windows above 20 packets, which is very small for modern networks.

This shows that AI can be both faster and less disruptive than MI.

### A. Very rapid increase

A more aggressive approach is to maintain the maximum rate of change, and simply reduce the time taken to increase. This results in increasing to the full fair window over a mere two RTTs, as done by MaxNet [3]. This corresponds to the dot-dashed line in Figure 1. Although less aggressive than RCP and QuickStart, which both increase to full window in a single RTT, this is too aggressive for schemes which merely guess the target window based on out-of-date information, such as after a TFRC idle period. The dangers include

- If the window is over-estimated, there is very little time to get feedback to indicate that.
- Although the maximum rate of increase is unchanged from straight doubling, it applies for longer. Thus, a competing long-RTT flow experiences a greater reduction in available bandwidth over one of its own RTTs.

## IV. Inaccuracy in target window

It is common for the target window $W$ to be either above or below the actual fair window $W_f$, for example as a result of changes in cross-traffic during the TFRC idle time.

Assume that the sender will receive a congestion notification one RTT after its window reaches the actual fair size. If $W > W_f$, then the window will overshoot. Assuming AI or MD is continued until the notification arrives, the overshoot will be proportional to the rate of change when the fair window is reached.

The proposed modification to TFRC [7] guards against overshoot by setting $W$ less than the window size at the start of the idle period. By making overshoot less severe, a less conservative value of $W$ can be used, potentially improving overall efficiency.

If the target window has been reached without receiving a congestion notification, the source must choose whether and how fast to continue increasing its window. For TFRC, a natural response is to continue with MI-2 as it does on its initial slow start. If QuickStart were modified to increase its window over a small number of RTTs instead of 1, a natural response would be to enter congestion avoidance once the window is reached.

## References

[1] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proc. ACM SIGCOMM*, London, UK, 1994, pp. 24–35.

[2] B. Wydrowski, L. L. H. Andrew, and M. Zukerman, "MaxNet: A congestion control architecture for scalable networks," *IEEE Commun. Lett.*, vol. 7, pp. 511–513, Oct. 2003.

[3] L. L. H. Andrew, K. Jacobsson, S. H. Low, M. Suchara, R. Witt, and B. Wydrowski, "MaxNet: Theory and implementation." [Online]. Available: http://netlab.caltech.edu/maxnet/MaxNet_ThImp.pdf

[4] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor sharing flows in the internet," in *Proc. International Workshop on Quality of Service (IWQoS)*, Passau, Germany, June 2005.

[5] S. Floyd, M. Allman, A. Jain, and P. Sarolahti, "Quick-start for tcp and ip," IETF, RFC 4782, Jan. 2007.

[6] M. Handly, S. Floyd, J. Padhye, and J. Widmer, "TCP friendly rate control (TFRC): Protocol specification," IETF, RFC 3448, Jan. 2003.

[7] E. Kohler, S. Floyd, and A. Sathiaseelan, "Internet draft draft-ietf-dccp-tfrc-faster-restart-03.txt (work-in-progress)," 2007.