Improving TCP Performance for Asymmetric Networks¹

Yi-Der Li and Wanjiun Liao

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
Email: wjliao@cc.ee.ntu.edu.tw

Abstract This paper proposes a new TCP variation called Formosa that significantly improves TCP end-to-end performance in terms of high connection throughput and high degree of fairness in resource sharing among multiple competing connections in asymmetric networks. Existing TCP implementations are based on ACK-clock, which clocks out a single data packet on receipt of an ACK. Formosa TCP pushes the frequency of ACKs per round trip time (RTT) to the lowest limit, using an ACK to cumulatively acknowledge a window's worth of data packets. The congestion window is updated to reflect the available bandwidth estimated on the forward channel, without the aids of the intermediate routers. As a result, the number of ACK packets is significantly reduced, while the principle of "conservation of packets" is still maintained. Based on "an ACK per window," Formosa estimates forward delay and implements the additive increase and multiplicative decrease algorithm for better congestion avoidance. In addition, it employs the negative acknowledgement mechanism for better congestion control during loss recovery. As compared to existing implementations, TCP Formosa demonstrates the best performance in terms of high connection throughput, high degree of fairness among connections and robustness in the face of two-way transfers in asymmetric networks.

Keywords: TCP, Asymmetric networks, Formosa TCP

1. Introduction

Congestion control in TCP has been an active research area since Jacobson attempted to solve the "congestion collapse" problem in 1986 [1]. Existing TCP implementations are based on the ACK-clock mechanism, using ACKs for senders to clock out new data packets into the network to protect networks from congestion.

Over the last two decades, TCP implementations have gone through many variations, including Tahoe, Reno and Vegas TCPs. In Tahoe TCP [1], during the slow start, the congestion window (cwnd) grows exponentially, doubling the cwnd per round trip time (RTT), until reaching a threshold (sssthresh), when congestion avoidance starts and the cwnd increases linearly. Reno TCP [2] enhances Tahoe with the [8][9][10], the authors have demonstrated that network asymmetry exacerbates this problem. [8] investigated the impairment of Reno TCP performance in many asymmetric scenarios. [9] studied how bi-directional traffic over asymmetric links negatively affects performance caused by ack-compression. [10] proposed some modifications to

inclusion of fast recovery, counting duplicate ACKs to estimate the number of outstanding packets during loss recovery. The growth of the Reno window in the equilibrium state follows the principle of AIMD (Additive Increase and Multiplicative Decrease), the stability of which has been widely demonstrated in the literature [3].

Vegas TCP [4] enhanced Tahoe and Reno with a better bandwidth estimation scheme for congestion avoidance, based on observing the variation of round-trip data transmission time so as to reflect the queue length in the bottleneck link, thus detecting congestion earlier rather than waiting until packet losses as in traditional TCP's. While enjoying the advantages of high throughput and low delay variation per connection, Vegas suffers from performance degradation when network bandwidth is asymmetric because its RTT estimation cannot identify the direction of the congestion. [5] suggested congestion avoidance algorithms based on monitoring delay and throughput variation and separating an RTT into a Forward Trip Time and a Backward Trip Time, and identifying in what direction a congestion occurs. In addition, a recent study on the trade-off between stability and fairness in TCP [6] proved that the congestion avoidance schemes which make the sender's cwnd converge to a fixed value as in Vegas may lead to unfairness when multiple active connections contend for bottleneck resources.

The ACK-clocked TCP implementations rely on the timely feedback of cumulative ACKs from the receiver to ensure reliability, to adjust the growth of the congestion window and to fully utilize available bandwidth in the forwarding path. Any disruption in the self-clocked feedback mechanism, especially one caused by severe ACK drops in the constrained reverse channel, may significantly degrade the performance on the forward data transfer [7]. Due to the cumulative property of acknowledgements, ACK dropping makes the sender put new packets into the network in bursts, thus disrupting the smooth ack-clocked transmission. In

endpoints and intermediate routers in Reno, including ACK Congestion Control (ACC), ACK Filtering (AF), ACK first scheduling and Sender Adaptation (SA), to alleviate the impact of ack-dropping in the reverse channel on performance. ACC with the aid of the Random Early Detection (RED) gateways and Early Congestion Notification (ECN) signals

¹ This work is supported in part by the National Science Council, Taiwan, under Grant Number NSC 89-2218-E-002-081 and Grant Number NSC 89-2219-E-002-030.

dynamically changes the delay-ack factor in receivers to reduce the number of ACKs in an adaptive way, while AF relieves ACK congestion by removing redundant ACKs per connection in the intermediate gateway's queue. ACK first scheduling gives ACK packets higher priority over data packets. SA eliminates sender burstiness and adjusts the *cwnd* according to the amount of data packets being acknowledged. All the existing solutions enhance performance in the presence of network asymmetry to some extent. However, in terms of high connection throughput and high degree of fairness among multiple simultaneous connections, these methods do not result in good performance on networks that exhibit asymmetry, because they are all based on the ack-clocked mechanism that generates too many ACK packets in the reverse channel. We wonder if ACK-clocked TCP were misbehaving or if they could be tuned to work better under network asymmetry. The answer to both questions is "yes."

This paper proposes a new TCP implementation called Formosa for asymmetric networks. Unlike existing TCP implementations based on ACK-clock, Formosa TCP pushes the frequency of ACKs per RTT to the lowest limit, using an ACK to cumulatively acknowledge a window's worth of data packets, rather than an ACK for a single data packet as in the ACK-clocked TCP. We will demonstrate how Formosa, based on "an ACK per window," provides congestion avoidance and control to achieve superior performance as compared to the existing TCP implementations, and verify our argument by simulation using the network simulator *ns*.

The rest of the paper is organized as follows. Section 2 describes TCP Formosa, including the new acknowledgement scheme, modified congestion avoidance and Negative ACK (NACK) for loss recovery. Section 3 evaluates the performance by simulation. Finally, the concluding remarks are included in Section 4.

2. TCP Formosa: protocol description

This section describes the techniques employed by Formosa in details.

2.1 Slow Start

The following example describes how the Formosa sender increases the *cwnd* during slow start based on "an ACK per window" to achieve "conservation of packets."

- Initially, $cwnd = 1$. The sender sends the first packet with the ack flag $A = 1$, indicating that the packet must be acked. The receiver acknowledges upon receipt of the 1st packet.
- Upon receiving the ACK in response to the 1st packet (for the 1st packet), the sender increases the *cwnd* by one (i.e., $cwnd = 1 + 1 = 2$) and sends the subsequent two packets: the 2nd packet with flag $A = 1$, and the 3rd packet with $A = 0$. The receiver acknowledges only on receipt of the 2nd packet.
- Upon receiving the ACK in response to the 2nd packet (for the 2nd packet), the sender increases the *cwnd* by one (i.e., $cwnd = 2 + 1 = 3$) and sends the subsequent three packets: the 4th packet with $A = 1$, the 5th with $A = 0$, the 6th with $A = 0$. The receiver acknowledges only on receipt of the 4th packet.

- Upon receiving the ACK in response to the 4th packet, which cumulatively acks the 3rd and 4th packets, the sender increases the *cwnd* by two (i.e., $cwnd = 3 + 2 = 5$) and sends the subsequent five packets: the 7th with $A = 1$, and the 8th, 9th, 10th and 11th with $A = 0$. Again, the receiver acks on receipt of the 7th packet only which cumulatively acknowledges the reception of the 5th, 6th and 7th packets.

The example above illustrates the growth of the *cwnd* during slow start, exhibiting the *Fibonacci* series. Let $cwnd(n)$ be the size of the *cwnd* in round n . Since the number of packets acknowledged by the ACK from round $n-1$ is equal to $cwnd(n-2)$, Formosa increases the window by $cwnd(n-2)$ packets, namely

$$\begin{aligned} cwnd(0) &= 1, \\ cwnd(1) &= 1, \\ cwnd(n) &= cwnd(n-1) + cwnd(n-2), n \geq 2. \end{aligned}$$

To eliminate the burst the sender may generate in response to an ACK, the Formosa sender spreads out the new data packets in time, scheduling the packets sent in the same round over an interval of $srtt/cwnd$. If no loss occurs, the Formosa receiver sends an ACK per RTT that cumulatively acknowledges unacknowledged data packets in the previous round.

2.2 Congestion Avoidance

Formosa enjoys the combined advantages of high throughput achieved by the estimation of forward-trip time variation (ftt_var) and the high degree of fairness of the AIMD algorithm. Upon receiving a congestion signal from the receiver, the sender reduces the *cwnd* by δ of its original value, $\delta < 1/2$, for the sake of achieving high throughput while maintaining fairness among multiple simultaneous connections. As a result, the window is reduced at a rate slower than the exponential decrease as in Reno when congestion continues, but faster than the linear decrement by one as in Vegas to enable slower senders to have the chance to grow their windows.

The behaviors of Formosa senders and receivers during congestion avoidance are summarized as follows.

Formosa receiver

- The Formosa receiver accumulates the increase of forward-trip time variation (ftt_var) for each data packet correctly received per RTT, namely,

$$ftt_var = \sum_{\substack{\forall \text{ data packet} \\ \text{received per RTT}}} (d - sftt) \quad (1)$$

where d is the forward delay of a packet received and $sftt$ is the smoothed forward delay of the packet in which $sftt = \eta * sftt + (1-\eta) * d$, $0 < \eta < 1$.

- Upon receipt of a packet with the flag A set, the Formosa

receiver determines if the sender should be informed, via the ACK sent in response, to shrink its window size. The throughput in this RTT is estimated as $packet_sum/srtt$, where $packet_sum$ is the amount of data received within an RTT interval and $srtt$ is the smoothed RTT. Note that $srtt$ is determined by the sender and piggybacked in the data packets to the receiver. As ftt_var cumulatively records the increase of forward delay variation since the previous ACK was sent, the extra packets L in the network generated in this RTT (i.e., queue length contributed in this RTT) is estimated as $L = ftt_var * packet_sum / srtt$. Let Q be the estimated extra packets in the network since the last congestion signal was sent to shrink the window size at the sender. $Q = \sum L$, $\forall L$ since the ACK with congestion signal was previously sent to shrink the cwnd. When Q reaches a threshold ($cathresh$), i.e., $Q > cathresh$, the Formosa receiver notifies the sender of congestion by piggybacking in the ACK, and then resets Q , i.e., $Q = 0$.

Formosa sender

1. On receiving an ACK packet with the congestion signal sent to shrink the cwnd, the Formosa sender multiplies its window size by a factor of δ , namely, $cwnd = (1 - \delta) * cwnd$, where $\delta < 1/2$.
2. The cwnd grows linearly by one per RTT in other conditions.

2.3 Loss Recovery

Formosa implements negative acknowledgements (NACK) to improve throughput during loss recovery. Once gaps (i.e., packet losses) in the data sequence have detected, the Formosa receiver stops sending ACK packets but sends NACK packets, which negatively acknowledge up to three non-contiguous *gaps detected* back to the sender to report losses, unlike SACK that positively acks up to three non-contiguous *blocks received*. A gap in NACK is expressed as a loss block, denoted $[a,b]$, where a and b are the two edges of the gap. A SACK packet is sent on receipt of an arriving data packet. A NACK packet is issued upon detecting a new gap by an arriving data packet. As in SACK, each NACK packet negatively acknowledges the gap most recently detected, plus two additional loss blocks that have been negatively acknowledged previously so as to increase the robustness of NACK packets. The advantage of repeating blocks to ensure reliability of NACK packets is particularly important in a network that exhibits asymmetry because the NACK packets may be constrained in the reverse link.

Figure 1. Simulation topology

3. Performance Evaluation

3.1 Simulation environment

This section describes the simulation setup for our experiments using network simulator *ns*. Fig. 1 depicts the simulation topology used, a network with bandwidth asymmetry. The degree of asymmetry is specified by a normalized asymmetry ratio r defined as the ratio of the transmission time of ACK packets in the reverse channel to that of data packets in the forward channel [9][10]. To measure the impact of reverse channels of different bandwidths on performance, we varied x from 20 Kbps to 600 Kbps, corresponding to a normalized asymmetry ratio from 20 to 0.67, respectively.

We made the following assumptions in our experiments:

1. Each data packet has size 1000 bytes, and an ACK packet, 40 bytes.
2. The routers have queue limit of 10 packets, in order to simulate the scenario of queue overflow in the intermediate routers.
3. The TCP clock granularity is set to 100 ms.
4. We use the First-In-First-Out (FIFO), Drop-Tail queuing discipline for the routers in Reno, SACK, Vegas and Formosa; but use RED gateway ($min_thresh=0.4$, $max_thresh=0.8$) with the ECN capability for the reverse bottleneck ($n2$ to $n1$) in ACC+SA.
5. Other parameters include: Vegas parameter $\alpha=1$, $\beta=3$, SA parameter $max_burst=1$, and Formosa parameter $\delta=1/8$, $cathresh=2$.

3.2 Simulation results

This section presents the simulation results on networks of different bandwidths in the reverse channel as shown in Fig. 1. We compare Formosa with Reno, ACC+SA, and Vegas. The scenarios we considered include (1) single one-way transfer to illustrate Formosa's behavior, and (2) multiple single one-way transfers to observe the fairness among multiple active connections. Note that we also simulated the scenarios of two-way transfers. The results are consistent with those of the other scenarios discussed below. Due to space limitation, we will not include those results in the paper.

3.1 Single one-way transfer

Fig. 2 depicts how asymmetry affects the throughput of each TCP implementation, with a normalized asymmetry ratio r , ranging from 0.67 to 4. r reaches one when the reverse channel has a bandwidth of 400Kbps, corresponding to the change over between symmetric and asymmetric bandwidths.

Figure 2. Throughput comparisons

As shown in Fig. 2, only Formosa is invariant to bandwidth asymmetry. The other TCP implementations reach their best throughput only when $r \leq 1$, when ACK packets would not be dropped thanks to sufficient resources in the reverse links. Even when $r \leq 1$, Formosa still performs the best, followed by Vegas, then by ACC+SA, and then by Reno.

The performance advantage of Formosa over the other implementations is particularly obvious when $r > 1$, thanks to its "an ACK per window," along with the modified congestion avoidance and loss recovery scheme tailored for scenarios with slow ACK paths. Reno achieves low throughput because it suffers from periodic, self-induced oscillation in the growth of the *cwnd* and the problem of multiple packet losses. In addition, when $r > 1$, Reno's ACK packets tend to congest and drop more severely in the reverse channel as r increases. On the average, only one out of r ACK packets can reach the sender. ACC is capable of reducing the amount of ACK packets, while companioned with the SA scheme sender burstiness and slow-down of window growth are alleviated so as to achieve better throughput as compared to Reno. While reducing portion of ACK, ACC could still cause ACK congestion when the reverse link is constrained. Vegas uses RTT estimation along with a fine-grained timer for better congestion avoidance to improve throughput. However, Vegas is very vulnerable to network asymmetry. Based on RTT estimation to adjust the *cwnd*, once a long round-trip delay is incurred by the ACKs in the constrained reverse channel, Vegas interprets the event as congestion in the forward direction and reduces the transmission rate accordingly, resulting in low throughput in the forward channel.

3.2 Simultaneous one-way transfers

In this experiment, we measured fairness and utilization among 10 active connections, all from n_0 to n_3 . Each connection starts at a time randomly selected from $[0,10]$ second. Our performance metrics are network utilization defined as the ratio of aggregate throughput of all competing connections to the maximum achievable throughput in the network [10] and the degree of fairness f defined as in [9][10],

$$f = \frac{\left(\sum_{i=1}^n x_i \right)^2}{n \sum_{i=1}^n x_i^2} \quad (2)$$

where n is the number of connections in the network, and x_i is the throughput of connection i , $1 \leq i \leq n$. The fairness index f is between 0 and 1: the larger the fairness index, the higher the degree of fairness.

Fig. 3 compares the performance of Formosa to Reno, ACC+SA, and Vegas when multiple active connections are contending for resources.

(a) Fairness

(b) Utilization

Figure 3. Multiple active connections

Fig. 3 (a) shows the fairness index plotted as a function of bandwidth in the reverse link, and Fig. 3 (b) shows the utilization of the four TCP implementations, with reverse link bandwidth varying from 20 Kbps to 600 Kbps. In both sub-figures, only the curves of Formosa stay high and flat across different bandwidth asymmetry scenarios, demonstrating that Formosa achieves high throughput and high degree of fairness among multiple active connections even in the presence of network asymmetry. The other implementations, when the reverse channel is constrained, do not allocate resources fairly among those competing connections, as shown in Fig. 3 (a). In Fig. 3 (b), both Reno and Vegas has very low throughputs when the normalized asymmetry ratio is large; ACC+SA is effective in resolving ack congestion; they appear more invariant to network asymmetry as compared to Reno and Vegas.

6. Concluding remarks

We have proposed a new TCP implementation called Formosa to achieve high throughput and high degree of fairness in resource sharing among multiple competing connections, even in the presence of network asymmetry.

Based on “an ACK per window,” Formosa estimates forward delay to implement the AIMD algorithm for better congestion avoidance, and implements the NACK mechanism for better congestion control during loss recovery. Compared to existing TCP implementations, Formosa gives the best performance in terms of high connection throughput and high fairness among connections in networks with symmetric or asymmetric links.

References

- [1] V. Jacobson “Congestion Avoidance and Control,” *Proc. ACM SIGCOMM '88*, August 1988
- [2] V. Jacobson “Modified TCP Congestion Avoidance Algorithm,” Technical report, Apr. 1990.
- [3] D. Chiu and R. Jain “Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks,” *Computer Networks and ISDN Systems*, Vol. 1, pp. 1-14, 1989.
- [4] L. Brakmo and L. Peterson “TCP Vegas: End-to-end Congestion Avoidance on a Global Internet,” *IEEE Journal of Selected Areas in Communications*, Oct. 1995.
- [5] O. Elloumi, H. Afifi, M. Hamdi “Improving Congestion Avoidance Algorithms for Asymmetric Networks,” in *Proc. IEEE ICC '97*, pp. 1417–1421.
- [6] G. Hasegawa, M. Murata, H. Miyahara “Fairness and Stability of Congestion Control Mechanisms of TCP,” in *Proc. IEEE INFOCOM '99*, pp.1329–1336.
- [7] L. Zhang, S. Shenker, and D. D. Clark “Observations and Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic,” *Proc. ACM SIGCOMM '91*, pp. 133-147, 1991.
- [8] R. Cohen “TCP for High Performance in Hybrid Fiber Coaxial Broad-Band Access Networks,” *IEEE/ACM*

Trans. On Networking, vol. 6, no. 1, Feb. 1998.

- [9] L. Kalampoukas, A. Varma, and K. K. Ramakrishnan “Performance of Two-Way TCP Traffic over Asymmetric Access Links,” *Proc. Interop '97 Engineers' Conference*, May 1997.
- [10] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz “The Effects of Asymmetry on TCP Performance,” *Proc. ACM MobiCom '97*.