Learning TCP - Adaptive Congestion Detection for Heteregenous Networks

Ajay Kumar Singh*, **Abhay Karandikar**, **Abhishek Jain**, **Shrinivas Kudekar** **and Sachin Katti**
Department of Electrical Engineering,
IIT Bombay, Mumbai, India.

Abstract— **Theory and experimentation have confirmed that as random drops increase in a network, TCP becomes inefficient and prone to instability due to its congestion control strategy. To address this problem we develop a novel approach to TCP congestion detection and recovery which outperforms TCP-SACK in heteregenous network environments, but still remains TCP friendly. The new Learning TCP (LTCP) which we propose, enables detection of random losses with a high degree of accuracy. LTCP does this by predicting the loss type when a loss is detected and dynamically updating the probability depending on correctness. Such knowledge of loss type enables it to avoid the throughput loss associated with TCP's normal recovery strategy.**

We demonstrate through extensive packet simulations that LTCP outperforms even TCP-SACK in environments where random drops dominate. We also demonstrate that LTCP is not unfriendly to legacy TCP implementations.

I. Introduction

FOR the Internet to continue to thrive, its congestion control strategy must remain effective and efficient as the network evolves. Current technology trends point to an increasingly heterogeneous network consisting of links of vastly different characteristics. Satellite networks, WLANS, optical links etc will be ubiquitous and an integral part of the future Internet. Since TCP was designed with the wired network in mind, its interaction with the new links might result in performance degradation and under utilisation.

Past studies [1] [2] have shown that in networks where packet drops are random and transient in nature, TCP's congestion control strategy will result in gross under-performance. TCP relies on packet loss as an indication of congestion and thereby lack of available bandwidth. Hence it reacts conservatively by either halving its sending rate or restarting from a window of 1. But this is obviously an incorrect reaction to random packet drops after which the link recovers. If such errors are frequent, TCP's performance degrades due to frequent fast retransmissions and consequent oscillatory behavior.

+Ajay Kumar Singh is with Tejas Networks, Bangalore, India.

Although the full impact of growing heterogeneity in the Internet on TCP's overall performance is yet to be experienced, we are already seeing seeds of it in TCP's performance over wireless networks. Bursty error conditions are relatively frequent in wireless channels, resulting in packet losses. The conservative reaction of TCP causes degradation in performance over these links which already do not offer very high bandwidth, making the problem more acute.

In this paper we propose a novel mechanism – Learning TCP (LTCP) – whereby we can detect the kind of loss when TCP detects a packet loss. LTCP classifies losses into two types - Type A and Type B - based on the recovery action it takes. Type A losses include normal congestion losses. Type B losses are the transient errors which heterogeneous networks introduce and which can be handled by a simple retransmission. LTCP's strategy is to make a prediction that the loss is a Type B loss, with a probability p_b, whenever a packet loss occurs. The probability is dynamically updated depending on correctness or incorrectness of prediction. Simulation results indicate that LTCP does a fairly good job of predicting the type of loss, thereby resulting in significant performance enhancement over existing TCP-SACK implementations.

The rest of the paper is divided as follows. Section II explains the design rationale, Section III describes the protocol, Section IV contains the simulation setup and results evaluating LTCP's effectiveness and we round off with conclusions and future work.

II. Design Rationale

Previous research [2] has shown that packet loss is not a reliable indicator of congestion on a heterogeneous network. As an implicit signal, loss is bad because congestion is not the only source of loss and also there is a delay in its detection. By the time we detect a loss on a heterogeneous network, the error conditions which were the source of the random loss might have ceased to exist, making TCP's reaction redundant. Hence we need methods to identify the type of loss and react accordingly.

Looking at the way the Internet is designed presently, we have no way of knowing explicitly the type of loss that occurred. Hence the sender has to make the best use of the only tools that it has at its disposal – packet loss and subsequent network reaction to one. If a Type A (i.e., congestion) loss occurrs, it is highly likely that more packets will be lost in the same window due to the congestion epoch. However, random drop(s) in the channel (Type B), will result in a packet(s) being dropped and the link being back to full capacity, unlike congestion.

Given the limited information available, predicting the type of loss seems to be the most prudent option. If a Type A loss is predicted, TCP's normal congestion control strategy takes over, else if a Type B loss is predicted, only the lost packets(s) is/are re-transmitted without taking any congestion avoidance action. Information about consecutive lost packets can be obtained by using TCP-SACK to implement LTCP. The correctness of the prediction can be ascertained easily since as we argued above, if we predicted a Type B loss and it turned out to be a Type A loss, subsequent losses would inform us about the incorrectness. The prediction probability should increase or decrease depending on correctness or incorrectness.

Though the prediction scheme seems to make sense, we should be careful not to overload the network. If we make an incorrect prediction, we are sending packets at our original rate to a congested network, during the time we are waiting to check the correctness of our prediction. Since any new mechanism will have to coexist with present TCP implementations, we might end up choking other TCP flows, if we are too aggressive with our predictions. Hence our heuristic of increasing prediction probability at every correct prediction should be applied conservatively.

III. The Protocol

LTCP is a design modification aimed at TCP implementations at end-systems. While we retain TCP's present congestion control strategy, we propose modifications in the overall congestion detection algorithms.

A. Congestion Detection and Prediction

TCP has two primary methods of loss detection – duplicate Acks (dupAcks) and time-outs. If a packet is lost and subsequent packets are successfully received by the destination, it generates dupAcks for the packet before the lost one. Such a situation occurs when there is transient congestion or if packet loss is due to transient error conditions in a heterogeneous network. TCP waits for three dupAcks to ensure that packet reordering is not accounted as packet loss. Type B losses can be detected by this mechanism. Hence when such a loss occurs, LTCP predicts whether the packet loss is of Type B or A.

After each transmission, TCP waits for a specific period which is a function of the RTT of the flow for an ACK. If neither an ACK nor 3 dupAcks are received, there is a timeout. Time-outs occur when the entire window of packets is lost, resulting in non-generation of even dupAcks and thus indicating periods of heavy congestion in the network. Hence if a packet loss is detected by time-out, LTCP does not make a prediction since it is likely to be wrong.

In the first case, LTCP predicts with a probability p_b, whether loss is of Type B. If it predicts that loss is of Type A, normal congestion control action is taken. But if the loss is predicted to be of Type B, only the lost packet is retransmitted and window size is kept at its present value, i.e., no congestion avoidance action is taken. Thus unnecessary window halving is avoided.

In wireless channels, errors are bursty in nature, resulting in consecutive packet losses in the same window. Since LTCP is implemented using TCP-SACK, dupAcks will be received with SACK information about packets received out of order. Hence LTCP will have exact knowledge about the number of consecutive packets lost. In such a scenario, LTCP retransmits all the consecutive packets lost.

B. Prediction Checking, Recovery and Probability Update

After a prediction is made it is necessary to check its correctness and update p_b accordingly. If the network is undergoing a congestion phase, it is highly likely that multiple packets in the same window will be lost. Let the TCP sequence number of the packet which is dupAcked thrice be `Last_Seqno` and the window when the prediction is made be `Win_At_Retrans`. Once LTCP makes a prediction that a loss is of Type B and retransmits the lost packet(s), it sets a timer – `Check_Prediction_Timer` – equal to $n * RTT$. If LTCP's prediction is right it should receive an ACK for a sequence number atleast equal to `Last_Seqno` + `Win_At_Retrans` within the `Check_Prediction_Timer` period.

In case the prediction is wrong, LTCP should be able to detect it as quickly as possible, otherwise it would be sending packets to a congested network at an unhealthy rate. Hence `Check_Prediction_Timer` should be set as low as possible. Simulations have shown that the optimum value for n is 1.5, given the variations in RTT estimates.

If the `Check_Prediction_Timer` times out, the prediction is wrong. Since a congestion state has been confirmed to be present in the network, LTCP reverts back to normal TCP congestion avoidance. Hence fast retransmission and recovery is initiated and p_b is decreased. Similarly if the prediction turns out to be right and an ACK is received for the whole window, p_b should be increased. The increase or decrease is based on the ratio of correct or incorrect to the total number of predictions, i.e., E_c or E_r. Since this might lead to high variance in the probability values, an Exponential Weighted Moving Average filter is used, to get more stable probability values , as given below and similarly for the decrease equation.

$$p_b = \alpha p_b + (1-\alpha)E_c \tag{1}$$

Receive (Packet P)
- **if** (newACK AND Check_Prediction) **then**
  - **if** ($P \rightarrow SeqNo \geq$ Prediction_Check_SeqNo) **then**
    - Cancel Check_Prediction_Timer
    - Check_Prediction = 0
    - Prob_Update (Increase p_b)
  - **end if**
- **end if**
- **if** (dupACK AND No_of_dupACKs == 3) **then**
  - **if** (Check_Prediction == FALSE) **then**
    - Predict with probability p_b whether loss is Type B
    - **if** (Prediction == Packet Loss is Type B) **then**
      - Retransmit one or consecutive lost packets
      - /* Consecutive lost packets can be found using TCP SACK */
      - Check_Prediction = TRUE
      - Prediction_Check_SeqNo = P $\rightarrow$ SeqNo + Present_Window
      - Set Check_Prediction_Timer
    - **else**
      - Normal TCP Congestion Control Action
    - **end if**
  - **end if**
- **else**
  - Normal TCP Action
- **end if**

Check_Prediction_Timeout()
- /* Timer timed out, hence prediction false */
- Check_Prediction = FALSE
- Normal TCP COngestion Control Action
- Prob_Update (Decrease p_b)

Algorithm 1: Window Control and learning P_b

C. *Throughput Calculation*

We calculate the throughput enhancement of LTCP over TCP. Let
p = Packet loss probability
p_a = Probability of type A loss given packet loss
p_b = Probability of type B loss given packet loss
p_l = probability of predicting that a loss is Type B
k = Number of packets acknowledged per TCP ack
Th_{tcp} = Throughput of TCP
Th_{ltcp} = Throughput of LTCP
E = Enhancement = Th_{ltcp} / Th_{tcp}

Packet loss is indicated by the reception of three dupAcks. It is assumed in this calculation that if a packet is lost, all other packets in that round are also lost. Between two triple-duplicate ACKs, the sender is in congestion avoidance phase, and the window increases by $1/k$ packets per round. Immediately after a loss the window is reduced to half it's size. As shown in [4], the throughput of the TCP source is approximately given by

$$Th_{tcp} = (1/RTT)\sqrt{3/2kp} \tag{2}$$

For TCP with random losses probability $p_b > 0$, the packet loss probability $p = p_a + p_b$, and hence,

$$Th_{tcp} = (1/RTT)\sqrt{3/(2k(p_a + p_b))} \tag{3}$$

Similarly for LTCP, we know that if it predicts random losses correctly, the window size is not reduced. Thus the packet loss probability p is given by $p = p_a$ since essentially in this case there is no throughput loss. If a random loss is predicted as a congestion loss, there is reduction in window and this amounts to a packet loss probability, $p_b(1-p_l)$. Also irrespective of the prediction, a congestion loss results in throughput loss and hence the throughput is given by the following equation,

$$Th_{ltcp} = (1/RTT)\sqrt{3/(2k(p_a + p_b(1-p_l)))} \tag{4}$$

From (3) and (4)

$$E = \sqrt{1 + (p_b \cdot p_l)/(p_a + p_b(1-p_l))} \tag{5}$$

In case of only wired networks ($p_b = 0$), and thus $E = 1$, i.e., LTCP achieves the same throughput as TCP and thus is TCP friendly. But in the presence of Type B losses, E will always be > 1.

IV. Performance

A. *Simulation Setup*

Our simulations use *ns-2* which we have extended with a LTCP module. The topology used is a multiple traffic, multiple link topology (see Figure 1) to simulate actual network conditions. The data

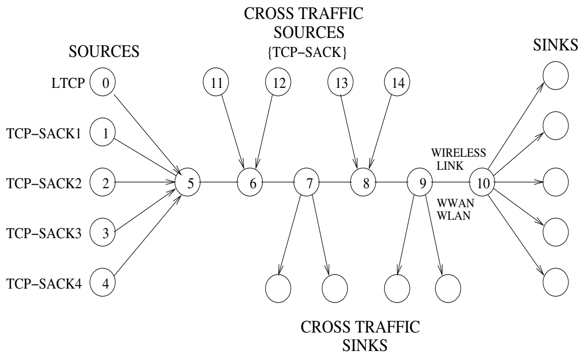

Fig. 1. Multiple Bottleneck Topology

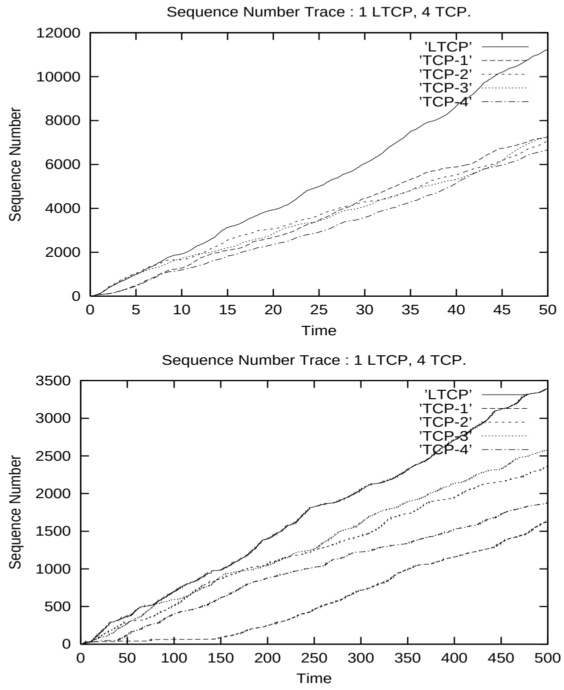

Fig. 2. Sequence No. trace for WLAN, WWAN

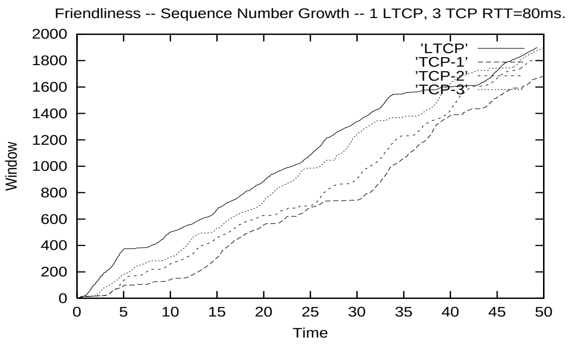

Fig. 3. LTCP's performance during heavy congestion

packet size is 1000 bytes. Bottleneck link capacity and round trip times are varied depending on the network being simulated (from 10 Mbps and 60 ms for WLAN to 200Kbps and 600 ms for Wireless WAN (WWAN)). Cross traffic TCP sources have smaller RTTs. Simulation times are always larger than 400RTTs and long enough for the simulations to reach a consistent state. We generate Type A and B losses in the following manner.

Type A Losses – Type A losses are induced by congestion caused by a constant bit rate (CBR) source sending packets at a high rate. Congestion naturally occurs at the bottleneck links resulting in dropping of packets. The sending rate of the CBR source can be thought of as a metric for the degree of congestion.

Type B Losses – To simulate random drops, we use a Two State Markov Error model which is inserted between routers 9 and 10 to simulate the heteregenous link. The link alternates between a good state and a bad state, dropping packets during the bad state with a certain probability which can be varied. The bad state duration is kept less than the RTT of the LTCP source so that drops occur within a single window. This is close to actual link conditions where error conditions are frequent but fleeting.

B. Simulation Results

Throughput Enhancement: We simulate WLAN and WWAN environments over the topology discussed above. LTCP is run with four other TCP-SACK sources with the same RTT, source destination pair etc. Random loss probability is set to 0.1. The sequence number traces in Figure 2 clearly illustrate that LTCP outperforms the other TCP-SACK sources significantly – to the tune of being able to send 40-60% more packets than the nearest TCP-SACK source.

TCP Friendliness: A CBR source is now introduced along with the LTCP and TCP-SACK sources. The CBR source transmits at 4 Mbps, which is enough to produce heavy congestion in the topology with a bottleneck link of 10 Mbps. As the Sequence number trace in Figure 3 illustrates, LTCP performs as well as the other TCP-SACK sources. In a heavily congested networks most of the losses will be due to congestion drops resulting in wrong predictions and consequent decrease in prediction probability. Thus LTCP will resemble a normal TCP-SACK source.

Prediction Correctness: We plot the `cwnd` trace for the LTCP source for both the WLAN and WWAN cases. If the prediction is correct, LTCP will recover from random losses without a window drop, else the window will be halved. As the plots in Figure 4 illustrate, LTCP is successful in recovering from many random drops without window halving. We also observe that some of the window drops have been due to multiple non-consecutive packet drops

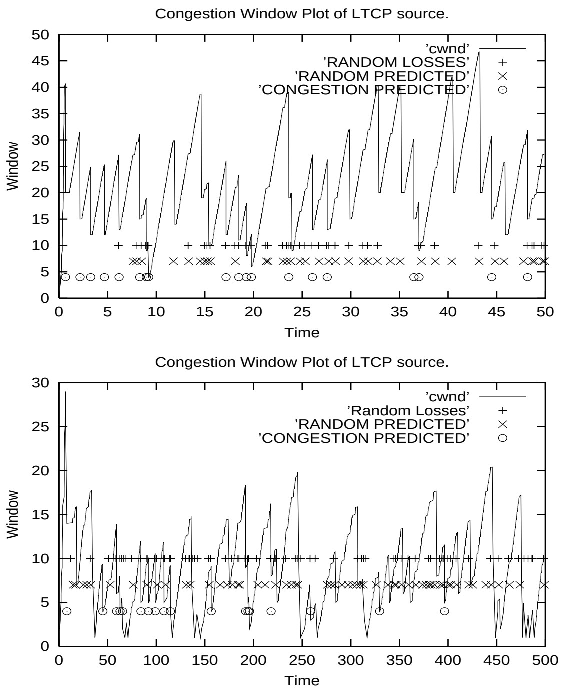

Fig. 4. `cwnd` trace for WLAN, WWAN

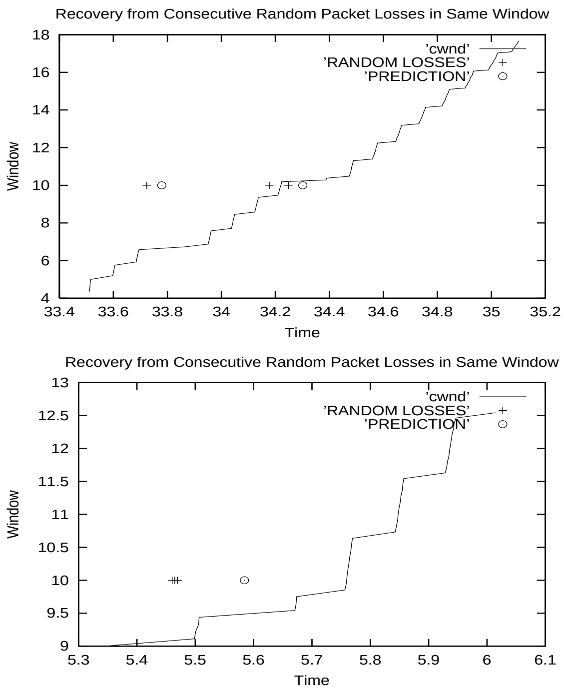

Fig. 5. Recovery from 2 and 3 consecutive random packet drops

in the same window. But in actual wireless channels this will be rare since losses will be bursty and consecutive and not spread over a window. Since LTCP can recover from consecutive packet losses, we believe that LTCPs performance will be even better in wireless networks.

Recovery from Consecutive Drops: We plot the `cwnd` trace for the period during which LTCP recovers from consecutive random packet losses in a single window. Figure 5 illustrates that LTCP recovers from 3 and 2 consecutive losses respectively without dropping the window. The SACKs received give information about these lost packets and LTCP recovers by retransmitting them immediately.

V. Conclusions and Future Work

In this paper we have presented a congestion detection mechanism for heteregenous networks. The salinet feautures of our scheme are

- LTCP identifes many random drops correctly and recovers without unnecessary window drops since the probability with which it predicts the type of loss is updated dynamically.
- LTCP recovers from consecutive packet losses since it uses TCP-SACK. This is a major advantage in wireless channels which have correlated losses.
- Since LTCP behaves as TCP-SACK in heavy congestion, it remains TCP friendly.

The limitation of current LTCP is that, it cannot recover from multiple random drops in the same window since SACKs can provide accurate information only about consecutive drops. This issue is currently being investigated.

VI. Acknowledgements

We would like to thank Mr Siddhartha Garg for his valuable inputs.

References

[1] Reiner Ludwig, Almudena Konrad and Anthony D. Joseph, "Optimizing the End-to-End Performance of Reliable Flows over Wireless Links", Fifth Annual International Conference on Mobile Computing and Networks (MobiCom '99), August, 1999, Seattle Washington.

[2] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan and Randy H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links", IEEE/ACM Transactions on Networking, 1997.

[3] Kevin Fall and Sally Floyd, "Simulation-based Comparisons of Tahoe, Reno and SACK TCP", Computer Communication Review, July 1996.

[4] Jitedra Padhye, Victor Firoiu, Don Towsley and Jim Krusoe, "Modeling TCP Throughput: A Simple Model and its Empirical Validation", ACM SIGCOMM '98 conference on Applications, Technologies, Architectures, and Protocols for Computer Communication.

[5] V. Jacobson, "Congestion Avoidance and Control", ACM Computer Communication Review; Proceedings of the Sigcomm '88 Symposium in Stanford, CA, August, 1988.

[6] The Network Simulator ns-2. http://www.isi.edu/nsnam/ns