Bayesian Piggyback Control for Improving Real-Time Communication Quality

Wei-Cheng Xiao and Kuan-Ta Chen

Institute of Information Science, Academia Sinica

Abstract—The critical tasks in designing a real-time communication system, such as online games, voice chat, and video conference applications, is to keep the end-to-end delay and jitter as low as possible. Delays and jitters are usually caused by network congestion and packet losses. A packet loss event will trigger the timeout retransmission mechanism at the transport layer, however, it is time consuming. Therefore, it is important to detect a loss event and do the retransmission efficiently. To achieve this goal, we propose a Bayesian loss detection mechanism based on the round trip time. We show that our loss detector can achieve high accuracy and high probability of detection as the false alarm ratio is lower than 20%. In addition, we also present a piggyback scheme for packet retransmission. Cooperated with the loss detector, the piggyback scheme can reduce the end-to-end delay and jitter to less than half of that of the original real-time communication system. Moreover, it generates few overheads to the network.

Index Terms—Bayesian inference, Online gaming, Parzen method, SCTP, VoIP

I. Introduction

Many real-time communication applications become more and more popular in recent years, including online games, voice chat, video conferences, etc. Traffic patterns of these applications are similar. They usually form small-size packets with high packet rates. To provide high interactivity, these applications require low end-to-end delay and jitter. A long end-to-end delay results from packet loss and the following timeout retransmission, and jitter is caused by network congestion or packet reordering. Jitter and long delay can reduce the smoothness of data transmission and produce lags during game play or voice chatting, which may degrade the user satisfaction. The real-time communication applications can be designed to detect loss events and then to do retransmission. However, there are some issues:

1) A good detector should work without modifying inherent network protocols and designs.
2) A detector has to judge whether a packet has been lost or not before the retransmission timer expires. It is too late if we wait until the transport layer agent triggers timeout retransmission.
3) The mechanism of packet retransmission after packet losses should be robust and efficient, and it should avoid generating too much unnecessary traffic.

To meet the first two issues above, we propose a loss detection mechanism based on Bayesian Inference. The *event* of a packet we are interested in contains two conditions, either *lost* or *successful*. We record observations of the RTT (round trip time) and loss events of each packet. Note that the RTT is the sum of propagation delay, transmission delay, processing delay, and queuing delay. However, it is the queuing delay that donates changes of RTT most. In a packet loss event, the last successfully transmitted packet is put at the end of a drop-tail queue and thus it will experience a high delay. Therefore, we can apply the RTT value and the event of each packet to the Bayesian Inference to guess events of a new coming packet based on previous observations. For each of the conditions, *lost* and *successful*, we construct two different conditional probability density estimators of RTT for applying the Bayesian Inference. One is the *histogram-bases approach*. The other is the *Parzon method*, in which the conditional probability density function is smoothened by modifying the histogram-based approach. Information needed for loss detection is previous observation only, and any additional probing is not required.

For the third issue mentioned above, we design a piggyback scheme on packet retransmission. In the piggyback scheme, payloads of lost packets are concatenated and then appended to a newly generated one. The piggyback function is triggered only if an event of packet loss is detected by our loss detector. This piggyback scheme helps decrease mean of end-to-end delay. Moreover, what's important is that it doesn't cause too much overhead.

We evaluate our framework using constant packet rate data with the network simulator *ns2*. During the process of performance evaluation, we analyze each packet at the end hosts and classify them as either *lost* or *successful* with our loss detector, and then performs the piggyback function when retransmission is considered necessary. In order to simulate an environment with network congestion, we inject some cross traffic into the network. In the simulation, we compare the two estimation techniques, the histogram approach and the Parzen method, with an optimistic and a pessimistic estimators under various congestion settings. The results show that our estimators achieve accuracy higher than 80% and that they help lower the end-to-end delay and jitter to 25%–50% of those of the original system. Moreover, our system has probability of detection higher than 80% as the false alarm ratio is lower than 20%.

The rest of the paper is organized as follows. We present the related works in Section II. Section III describes the Bayesian Inference and our loss detection mechanisms and the piggyback scheme. The performance evaluation and its results are shown in Section IV. Finally, we conclude our paper in Section V.

II. RELATED WORK

The packet loss detector used in this paper adopts the ideas of a Bayesian packet loss detector for TCP [2]. We also use the two estimation techniques, the histogram-based approach and the Parzon method, and run packet loss event detection based on the observation of RTT values. However, what we focus is whether a packet is *lost* or *successfully* received in real-time communication systems, while in [2] they emphasize packet loss and reordering for TCP connections. Moreover, we propose a piggyback scheme cooperating with the loss detector for data retransmission.

There are many redundancy mechanisms for loss recovery in delay constrained systems, which include real-time audio data transmission [1, 3–6, 8, 11], and real-time video delivery [9, 10]. Kouvelas *et al.* [6] uses a piggyback scheme similar to ours. Moreover, they focus on controlling the amount of redundancy under different network conditions using an intelligent probing technique. Tong *et al.* [11] proposes a four-step mechanism, with piggyback and erasure coding included, to recover multiple and consecutive packet losses. Rather than implementing redundancy at packet level, some coding schemes are also applied in [5, 12], such as parity coding and Reed-Solomon coding, to achieve data recovery. A disadvantage of coding-based loss recovery mechanisms is that more CPU resources are required for decoding. Like our piggyback scheme, in [8–10], a packet may encapsulate multiple payloads, including new and redundant data. This simple packetization method helps reduce additional packet headers. There is also a survey proposed by Perkins *et al* [7]. They emphasize streaming audio on IP-multicast network and divide the recovery schemes into two classes, sender-based recovery and receiver-based recovery. Moreover, they give some comparisons and discussions on the recovery schemes of the two classes.

III. BAYESIAN PIGGYBACK CONTROL

In this section we give an introduction to the Bayesian Inference mechanism. Then, we propose two estimation techniques for our packet loss detector using the Bayesian frameworks based on the distribution of round trip time.

In the Internet, packets are transmitted via intermediate routers. Packet round trip time is summed up from the processing delay, transmission delay, propagation delay, and queuing delay. Among these four kinds of delays, it is queuing delay that mainly donates the round trip time. We assumes drop-tail queues in the following works. In a drop-tail queue, a new incoming packet will be dropped by the router when the queue is full. The packet preceding it will experience the longest queuing delay, and a long round trip time consequently. Therefore, we can detect packet loss events by observing the RTT value of each successfully delivered packet.

A. Bayesian Inference

Bayesian Inference is based on the Bayes' Theorem :

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)},$$

where $\Pr(B|A)$ is called the *likelihood*, $\Pr(A)$ is the *prior*, and $\Pr(A|B)$ is the *posterior*. The posterior probability is the probability of the *hypothesis B* conditioned on the *evidence A*. While applied to the Bayesian Inference, it becomes

Posterior ∝ *Likelihood * Prior.*

In our works, we use a two-state hypothesis, *lost* and *successful*, which are the packet events we are interested in. We take the RTT value as the *evidence* for inferencing the packet events. Let L denotes the set of packet events {*lost*, *successful*}, and R represents the RTT value. Then, we can rewrite the generic Bayesian Inference :

$$\Pr(L|D) \propto \Pr(D|L)\Pr(L),$$

where D denotes the distribution of RTT values. Once we have a new record of RTT value and packet event from the observation, we update the likelihood and prior to renew the distribution of posterior probability. We can guess the event of a packet not acknowledged yet by following the steps:

1) Calculate the difference of the current time and the timestamp when the packet is sent out. The time difference is denoted d_1.
2) Figure out the posterior probabilities $\Pr(lost|d_1)$ and $\Pr(successful|d_1)$.
3) Multiply the two posterior probabilities above by *penalties*, which will be introduced later. After that, we can get two *scores*, S_{lost} and $S_{successful}$.
4) Compare the two scores and choose the event with higher one. If the results indicate that the packet is more likely to be lost, we trigger packet retransmission using the piggyback scheme, which will be discussed in Section III-C.

Along with accumulation of evidence, the degree of belief in the hypotheses changes. With enough evidence, the distribution of posterior probability will get more closer to the real situation.

B. Probability Density Function Estimation

Before applying the Bayesian Inference to our loss detector, we need a way to estimate the conditional probability densities for the two hypotheses *loss* and *successful*. Here we employ two techniques, a histogram-based approach, and a more elegant technique using the Parzon method.

1) Histogram: This is a simple method to estimate the probability mass function. The scope of RTT measurements is divided into multiple commensurate *bin*s, i.e. $\{b_1, b_2, ..., b_n\}$. For all the RTT intervals $b_i, i = 1, 2, ..., n$, we maintain a set of count $\{c_1, c_2, ..., c_n\}$. Every time a RTT value observed, the count c_i is increased by 1 if the observed RTT value falls into the interval b_i. The probability is simply:

$$p_i = \frac{c_i}{\sum_{j=1}^{n} c_j}, \text{ for } i = 1, 2, ..., n.$$

2) The Parzen Method: The Parzen method estimates the probability density function by applying a kernel function $K(\cdot)$

on each sampling point of evidence. In our work, we use the Gaussian kernel:

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}.$$

While curves of the naive approach are histograms, the Parzon kernel helps smooth the curves of likelihood function such that we can evaluate the points of interest where we don't have any evidence yet. After applying the kernel function, the probability density function becomes:

$$p(r) = \frac{1}{bN\sqrt{2\pi}} \sum_{y \in S} e^{\frac{-(r-y)^2}{2b^2}},$$

where r is the RTT value of interest, b is the Parzen bandwidth, and S is the set of N RTT measurements.

We test the hypotheses using the Bayesian Inferences mentioned above. However, if the guess goes wrong, negative impacts of the two, *lost* and *successful*, are quite different. The two cases of wrong judgement are denoted *false positive* (or *false alarm*) and *false negative* respectively:

> *false positive*: an event *successful* is judged as *lost*;
> *false negative*: an event *lost* is judged as *successful*.

In our work, a false alarm will cause the end host to retransmit the packet that is misjudged lost, and some additional traffic will be injected into the network. However, it might be worse if a false negative occurs. In this case, the packet is really lost, but this event will not be detected and the packet will not be retransmitted by our framework. It will cause a very high end-to-end delay and affect user experiences seriously. To avoid this condition, we add the idea of *penalty* to the posterior probabilities while making judgements on the events of a packet. Penalty of the hypothesis *lost* is set higher than that of the hypothesis *successful*. While making the judgement, we compare the two following *scores*:

$$S_{lost}(r_0) = p_l \cdot \Pr(lost|r_0),$$

$$S_{successful}(r_0) = p_s \cdot \Pr(successful|r_0),$$

where p_l and p_s ($p_l \geq p_s$) are penalties of hypotheses *lost* and *successful* respectively, and r_0 is a sample of time difference we measured. These penalties can help decrease the false negative rates with a few increases of false positive rates.

C. Piggyback Scheme

After our detector makes judgements, a retransmission mechanism should be triggered if a packet is considered lost. We propose a *piggyback scheme* for application level data retransmission. In the piggyback scheme, each time a new message is generated, previous data considered lost will be appended to construct a new packet. The piggyback scheme helps retransmit lost data before the transport layer timer expires. Moreover, only before new data are to be sent will the lost data be appended for retransmission. Compared with general retransmission schemes, our piggyback scheme reduces the bandwidth requirement for packet headers and then decreases network overheads.

IV. PERFORMANCE EVALUATION

In this section we show the performance of the two estimators of our loss detector mentioned in Section III-B, the histogram-based approach and the Parzon Method. We run a series of constant packet rate data on the network simulator ns2 to simulate a real-time communication system. In addition, we introduce two simple mechanisms for comparison purpose. One is the original system without any loss detection or user-level retransmission mechanism. The other is a mechanism in which a message will be always retransmitted until it is successfully received. We name these two mechanisms *optimistic* and *pessimistic* in later sections.

A. Simulation Setup

We simulate a real-time interactive communication system on the simulator ns2. In our simulation, a 50-node network topology is generated to form a transit-stub graph. This topology contains one transit domain and six stub domains. Each transit domain has two nodes, and each stub domain has eight nodes on average. The average bandwidth of transit-transit, transit-stub, stub-stub domains are 2000 KB/sec, 2000 KB/sec, 1000 KB/sec respectively. Among these nodes, one of the transit nodes is assigned as the communication server, which is responsible for data broadcasting and transporting. Fifteen of others are general hosts with real-time communication client applications running on them. The rest nodes are hosts generating cross traffic, which is used to interfere with the normal communication traffic by producing effects of network delay and packet loss.

The real-time communication application generates data with a constant packet rate every 30 ms a packet. The packet size follows a uniform distribution between 100 and 300 bytes. The hosts running the client applications send data to and receive data from the communication server following the packet rate and packet size settings above. We select SCTP as the transport layer protocol in all our simulations, and it is set to be *connection-oriented*, *reliable transport*, and *unordered delivery*. We manually deal with the ordering problem inside the communication application. During the simulation, UDP packets are injected into the network to be the cross traffic. These UDP packets are sent out by 0–10 pairs of hosts different from those hosts running the real-time communication application. The overhead donated by each host is around 750 KB/sec. Besides, the penalties in the score functions are set to 1 and 10, 000, 000 for the hypotheses *successful* and *lost* respectively.

B. Performance of Loss Detector

In this section we show the probability of detection and the accuracy of our loss detector. In Fig.1 we plot the accuracy curves of our loss detector. We can see that the Parzon method always performs better than the histogram-based approach under all cross traffic settings. The high accuracy means that our detector keeps the false positive rate and false negative rate low and that it doesn't inject too much useless traffic into the network. For the two mechanisms, *optimistic* and *pessimistic*,
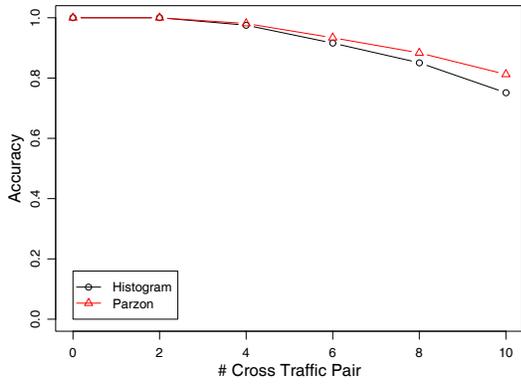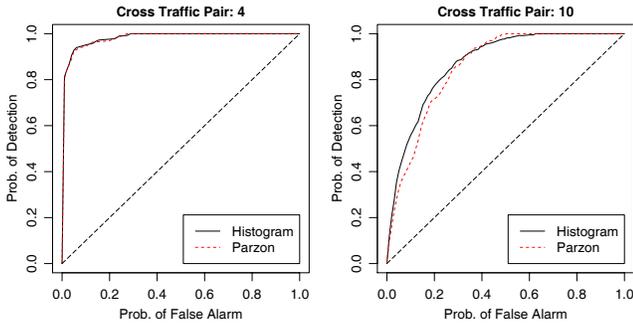
Fig. 1. Accuracy

Fig. 2. ROC Curve

Fig. 3. E2E Delay of different delay bounds

Fig. 4. E2E Delay of different number of cross traffic pairs

the ROC and accuracy curves are not shown here because we don't apply our detector on them. They are for performance comparison only.

In Fig. 2 we plot ROC curves of the histogram-based approach and the Parzon method under different number of cross traffic pairs and various training settings. We can see that when the false positive rate is 20%, the probability of detection ranges from 80% to near 100%. These ROC curves indicate that our detector is quite accurate and reliable.

C. Effects of Piggyback Scheme

In a real-time interactive communication system, end-to-end delay directly reflects user satisfaction. Long end-to-end delays will cause users to quit a system. Another factor that influences user behavior is the time difference of two contiguous message generated by the applications. The time difference is denoted *lag* in our work. The lower value of lag a system has, the more smoothly the system performs.

Results of end-to-end delay analysis are shown in Fig. 3 and Fig. 4. From these charts we can see that both the histogram-based approach and the Parzon method perform much better than the optimistic mechanism. In the results of these two techniques, most end-to-end delay values are below 1 sec, and the probability is higher than 99%. The *pessimistic* mechanism surely performs the best on the delay because it always retransmit old messages. However, as shown in Fig. 7, it also induces more than twice as many overheads as other mechanisms do. Therefore, it is not a feasible solution in real world network.
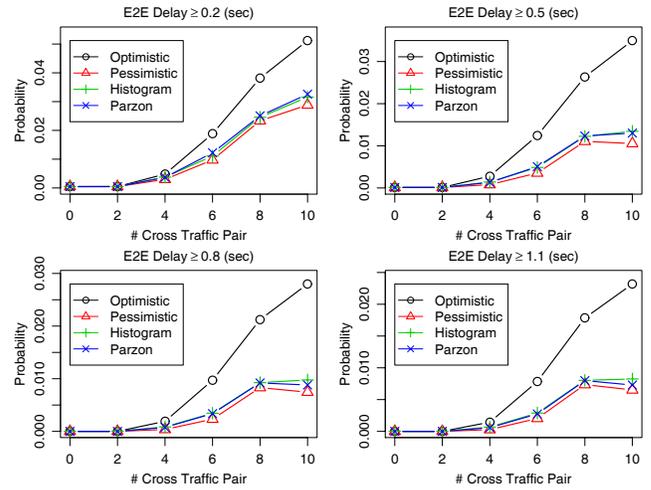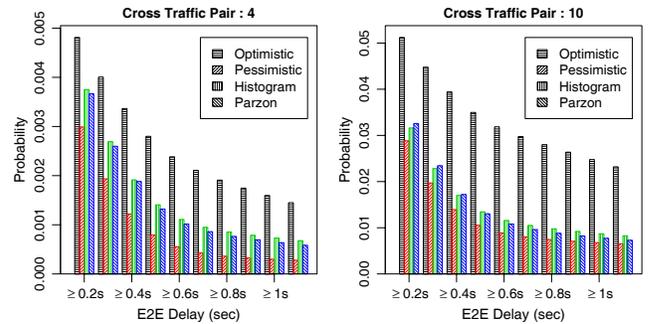
Figure 5 and Fig. 6 show the performance comparison results about lag. Similar to the results of end-to-end delay, both the histogram-based approach and the Parzon method achieve lags about only 25%–50% of those of the optimistic mechanism. It implies smoothness of the system with our detector built-in.
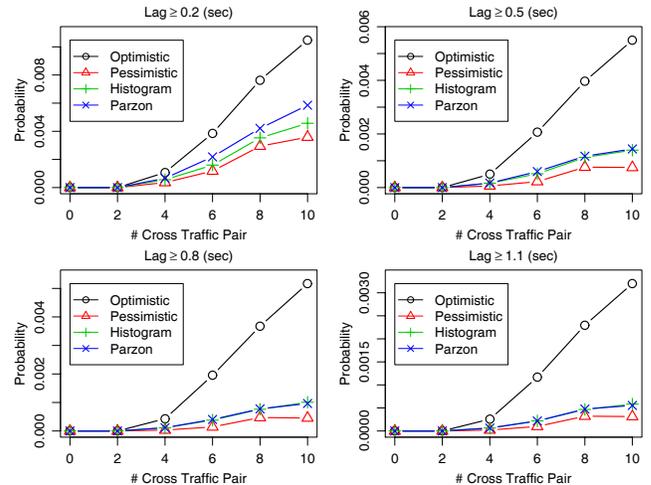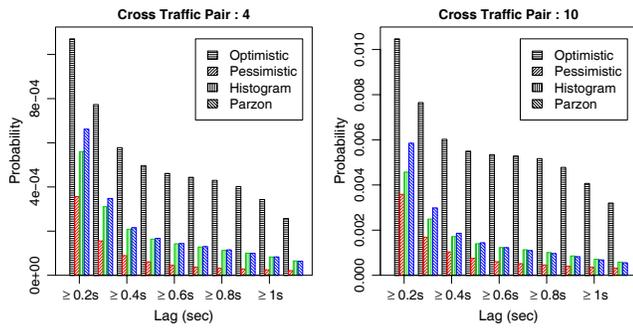

Fig. 5. Lag of different lag bounds

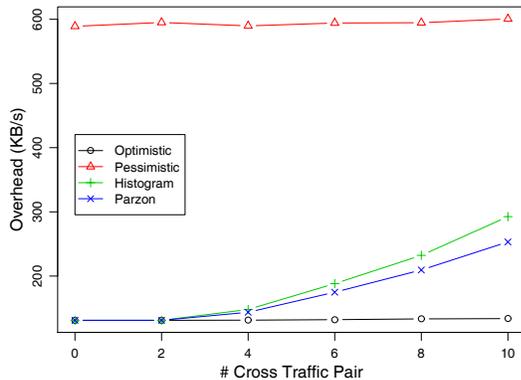Fig. 6. Lag of different number of cross traffic pairs

Fig. 7. Overheads

V. CONCLUSION

In this paper we give a brief view of the Bayesian Inference mechanism. Then, we introduce our detector using the Bayesian frameworks to judge whether a packet is lost or not. Our loss event detector works according to records of round trip time of the preceding packets. We also propose two techniques, the naive approach and the Parzon method, to assist the Bayesian Inference scheme. Both techniques use weighted probability function to make judgement between the hypotheses *lost* and *successful*. The weights help decrease false negative rates and shorten the end-to-end delay.

Through the performance evaluation, we show that our two techniques achieve good performances on end-to-end delay, lag, and the ROC curves. Both the techniques can detect loss events with high accuracy, and the Parzon method is even better. From the figure of end-to-end delay and lag, we can see that our estimators behave at least twice better than the *optimistic* mechanism does, and that it is only a little worse than *pessimistic*. However, overheads induced by our estimators are much fewer. Through the ROC curves we find that our detector achieves a probability of detection above 80% as the false alarm is below 20%.

REFERENCES

[1] J. Bolot, A. Vega-Garcia, and S. Inria, "Control mechanisms for packet audio in the Internet," *Proceedings of IEEE INFOCOM'96*, vol. 1, 1996.

[2] N. Fonseca and M. Crovella, "Bayesian packet loss detection for TCP," in *Proceedings of IEEE INFOCOM 2006*, May 2006.

[3] T.-Y. Huang, K.-T. Chen, and P. Huang, "Tuning the redundancy control algorithm of Skype for user satisfaction," in *Proceedings of IEEE INFOCOM 2009*, April 2009.

[4] T.-Y. Huang, P. Huang, K.-T. Chen, and P.-J. Wang, "Can Skype be more satisfying? – a QoE-centric study of the FEC mechanism in the internet-scale VoIP system," *IEEE Network*, 2010.

[5] W. Jiang and H. Schulzrinne, "Comparison and optimization of packet loss repair methods on VoIP perceived quality under bursty loss," in *Proceedings of ACM NOSSDAV 2002*, 2002, pp. 73–81.

[6] I. Kouvelas, O. Hodson, V. Hardman, and J. Crowcroft, "Redundancy control in real-time Internet audio conferencing," *Proceedings of AVSPN*, vol. 97, 1997.

[7] C. Perkins, O. Hodson, and V. Hardman, "A survey of packet loss recovery techniques for streaming audio," *IEEE Network*, vol. 12, no. 5, pp. 40–48, 1998.

[8] C. Perkins, I. Kouvelas, O. Hodson, V. Hardman, M. Handley, J. Bolot, A. Vega-Garcia, and S. Fosse-Parisis, "RTP payload for redundant audio data," *Request for Comments (Proposed Standard)*, vol. 2198.

[9] M. Seo, Y. Jeong, K. Seo, and J. Kim, "Piggyback packetization of duplicate packets for packet-loss resilient video transmission," *IEICE Transactions on Communications*, vol. 89, no. 10, p. 2802, 2006.

[10] M. Seo, Y. Jeong, J. Kim, and K. Park, "A new packet loss-resilient duplicated video transmission," *Asia-Pacific Conference on Communications 2005*, pp. 1063–1067, 2005.

[11] K. Tong and H. Kong, "Lightweight piggybacking for packet loss recovery in Internet telephony," in *Proceedings of IEEE ICC 2007*, June 2007.

[12] S. Yuk, M. Kang, B. Shin, and D. Cho, "An adaptive redundancy control method for erasure-code-based real-time data transmission over the Internet," *IEEE Transactions on Multimedia*, vol. 3, no. 3, pp. 366–374, 2001.