

# Cross-Layer Speculative-Parallelization Protocol Architecture for End Systems and Gateways in Computer Networks With Lossy Links

**Haowei Bai**

Space Applications  
Honeywell Aerospace  
19019 N. 59th Ave., Glendale, AZ 85308  
haowei.bai@honeywell.com

**David J. Lilja**

Electrical and Computer Engineering  
University of Minnesota  
200 Union St. S.E., Minneapolis, MN 55455  
lilja@umn.edu

**Mohammed Atiquzzaman**

School of Computer Science  
University of Oklahoma  
Norman, OK 73019-6151  
atiq@ou.edu

**Abstract**—The throughput degradation of Transport Control Protocol (TCP)/Internet Protocol (IP) networks over lossy links due to the coexistence of congestion losses and link corruption losses is very similar to the degradation of processor performance (i.e., cycle per instruction) due to control hazards in computer design. First, two types of loss events in networks with lossy links are analogous to two possibilities of a branching result in computers (taken vs. not taken). Secondly, both problems result in performance degradations in their applications, i.e., penalties (in clock cycles) in a processor, and throughput degradation (in bits per second) in a TCP/IP network. This has motivated us to apply speculative techniques (i.e., speculating on the outcome of branch predictions), used to overcome control dependencies in a processor, to throughput improvements when lossy links are involved in TCP/IP connections. The *objective* of this paper is to propose a cross-layer system to improve the network throughput over lossy links. This system consists of protocol-level speculation based algorithms at transport layer, and protocol enhancements at middleware and network layers that provide control and performance parameters to transport layer functions. Simulation results show that, compared to prior research, our proposed system is effective in improving network throughput over lossy links, capable of handling incorrect speculations, fair for other competing flows, backward compatible with legacy networks, and relatively easy to implement.

**Keywords:** TCP, Congestion control, Explicit congestion notification (ECN), Speculative execution, Wireless networks

## I. INTRODUCTION

Networks with lossy links, such as RF wireless networks, have a number of characteristics inherently different from wireline networks, for which the TCP/IP protocol suite was originally designed. Notable among them is the transmission error measured by bit error rate (BER). Few errors per packet may be corrected by lower layer encoding schemes. However, more errors may result in packet drops. Discarded packets will not be handed up to applications; therefore it is the responsibility of transport layer protocols to handle such issues. The original TCP protocol uses packet losses as indications of network congestion, because congestion losses were the only losses that are not completely detectable and correctable in wireline networks. However, in a network with lossy links (include RF wireless links, IR wireless links, satellite links, and even copper and fiber links exposed to high-dose rate radiations), packet losses due to link errors are not related to network congestions. Unfortunately, the current TCP treats these losses as congestion losses, and in turn reduces the

transmission speed. This issue is currently the subject of much debate and research in the networking community, and has led to the development of enhancements.

Despite of much efforts in the past several years, none of the proposed enhancements, ranging from algorithms at the link layer, and middleware layer, to algorithms across multiple layers, have provided compelling evidence that they are suitable for mass deployment. Designing a network protocol satisfying a set of design requirements has been difficult to achieve. A set of good criteria to measure the proposed algorithms has yet to be agreed by the networking community. Consequently, this still remains as an open issue. Nevertheless, any new networking protocols should meet certain basic requirements.

The *objective* of this paper is to investigate the feasibility of, and furthermore, evaluate the performance of, applying speculative parallelization to the design of protocols that improves network throughput over lossy links. To achieve this goal, we first will establish a number of basic requirements that a new algorithm for this purpose should possess. We then blueprint a system architecture integrating protocol enhancements at middleware, transport, and network layers, which can meet all these requirements. With this road map in mind, we will describe the function and principle of each functional block, and present performance evaluation results.

The systematic design methodology described in this paper, i.e., design and validate against clearly established requirements, orchestrates the overall research work, which has made the following contributions:

- A cross-layer system is developed to improve the network throughput over lossy links. This system consists of protocol enhancements at middleware, transport, and network layers. Enhancements at the middleware layer provide control and management functions, and enhancements at the network layer provide performance parameters to protocol-level speculative parallelization algorithms at the transport layer. We have analyzed the feasibility of this system and found that it is effective in improving network throughput over lossy links, capable of handling incorrect speculations, fair for other competing flows, backward compatible with legacy networks, and relatively easy to implement.
- We propose a protocol-level speculation algorithm to im-

*prove the network throughput over lossy links.* This transport layer algorithm consists of a conditional Bernoulli predictor used to predict the type of a loss event, and a speculative congestion control algorithm used to appropriately adjust the window size for the predicted loss events. Our simulation evaluation results have shown that the conditional Bernoulli predictor needs less memory space but produces higher prediction accuracy than other previous network predictors. The proposed speculative congestion control algorithm outperforms several previous TCP improvement algorithms as well as the baseline TCP-Reno.

- *To maximize the speculation accuracy, we develop mathematical models to minimize congestion losses at ECN-capable random early drop (RED) gateways, by optimally dimensioning the maximum buffer size and the maximum threshold at the network layer for ECN-capable RED gateways.* This is critical to the function of conditional Bernoulli predictor at the transport layer. Speculative techniques are used to maximize the instruction-level parallelism for modern processor design, at the price of the need of additional hardware to handle incorrect speculations. However, in the problem we aim at, a speculation result is either congestion loss or link corruption loss. If we knew the speculation was incorrect, i.e., we knew which type of loss it was, the problem of the coexistence of two types of losses would not need to be formulated. Therefore, our focus is to improve the speculation accuracy, without adding too much complexity to do it. Simulation-based verification results have shown that our mathematical models can effectively adjust RED gateway parameters so that congestion losses can be minimized. Most significantly, the buffer size inside a RED gateway dimensioned by our models are much smaller than previously suggested values. This means that, without other contributions in this paper, the modeling task itself has significant contribution to the network performance improvement.

The rest of this paper is organized as follows. In Section II, we explain why and how we apply speculation parallelization to network protocol design. The proposed system architecture is presented in Section III. The proposed protocol-level speculation algorithm is described in Section IV. In order to improve the speculation accuracy, we describe our mathematical models used to minimize congestion losses in Section V. System performance evaluation results are presented and analyzed in Section VI. Finally concluding remarks are given in Section VII.

## II. THOUGHT PROCESS: FROM PROCESSOR DESIGN TO NETWORK PROTOCOL DESIGN

In this section, we describe our thought process, which leads to the main idea of this paper. There have been many algorithms proposed to improve network throughput in wireless

environment. Due to the limited space, please refer to our previous work [1] for a more detailed review of previous work.

### A. Improving Instruction Level Parallelism By Speculations

One of the major contributions to modern processor design is the use of pipelining architecture to overlap the execution of instructions, thereby improving the performance. Accordingly, all techniques used to increase the amount of instructions per clock cycle at this level are called instruction-level parallelism (ILP) techniques. For typical MIPS programs, the average dynamic branch frequency is between 15 to 25 percent [2]. Since this type of instructions often depends on each other, the amount of parallelism to be exploit is much less than the average blocks without branch instructions.

Determining how one instruction depends on others is essential to ILP. One type of dependency related to branches is called control dependency. The control dependency cannot be removed, i.e., an instruction control-dependent on a branch cannot be moved to a place before the branch instruction. In addition, an instruction that is not control-dependent on a branch cannot be made dependent, i.e., this instruction cannot be moved into the branch. This poses the limitation on the flexibility of dynamic scheduling and/or branch delaying. Such limitations result in control stalls (one to several clock cycles) waiting for branch result(s) so that instruction stream can be resumed. Branch prediction was proposed to predict the branch results before the execution of branch instructions is completed. This reduces control stalls at the price of incorrect predictions. The simplest branch prediction is a branch prediction buffer or branch history table that contains a bit indicating whether the recent branch result was taken or not, based on which, future branch results are predicted so that a continuous instruction stream is provided to the processor.

A more advanced scheme called hardware based speculation is used for processors executing multiple instructions per clock cycle to maximize the parallelism. The hardware based speculation combines dynamic branch prediction, speculation allowing instruction execution before branches are resolved (out-of-order execution, and in-order commitment), and dynamic scheduling of blocks. The most obvious disadvantage of hardware based speculation is the complexity and additional hardware. A software speculation at compiler called static branch prediction was proposed for branch behaviors that are highly predictable at compiler time. Such predictions are often helpful for scheduling data hazards.

Having reviewed the basics of speculative techniques that are used by computer architects for processor design, our purpose is to lay a ground for the rest of this paper, in which we will describe the existence of similarities between processor design and the design of some network protocols. Furthermore, we will show similar ideas can be applied to protocol design to improve the network performance in the environment creating the similar issues.

## B. Crosscutting Question: Is Speculation Helpful to TCP/IP Networks?

TCP was originally designed for wireline networks, where packet losses are mostly caused by network congestions. The current TCP algorithm uses either retransmission timer timing out, or receipts of three duplicated acknowledgements (ACKs) sent by receivers, to implicitly indicate loss events. However, wireless links are characterized by high error rates (see [3] for a tutorial on errors in wireless networks). In most cases, packet losses due to corruption are more significant than congestion losses when a wireless link is involved in a TCP connection. In such a case, TCP may not be able to transmit or receive at the full available bandwidth, because the TCP algorithm will be unnecessarily wasting time in slow-start or congestion avoidance procedures triggered by link errors [4]. Consequently, the current congestion control algorithms in TCP result in very poor performance over wireless links. It is expected that a modification could be made which will enable the TCP congestion control algorithm to differentiate, and furthermore behave appropriately in the presence of congestion and corruption losses. Significant performance improvements can be achieved if losses due to network congestion and corruption in lossy wireless links could be appropriately differentiated [5].

Similar problems exist in computer architecture design, where control hazards prevent a processor from starting the next instruction before the correctness of branch prediction results is verified. This may cause at least one-cycle penalty which degrades the processor performance. Researchers try to overcome control dependencies in order to exploit more instruction-level parallelism by speculating on the outcome of branches and executing the program as if predictions were correct [6] [2]. Meanwhile, it is necessary to have a scheme with ability to handle the situation where the speculation is incorrect (e.g., flush the reorder buffer). The similarity between the degradation of processor performance due to control hazards in computer design, and the degradation of network throughput due to the coexistence of two types of losses in network protocol design is summarized in Table I. If we could eliminate the waste of bandwidth responding to link errors, using speculative techniques as computer architects would do to eliminate the waste of time waiting for branching results, we would significantly improve the network throughput.

In order to improve the TCP throughput over lossy links, we propose to use a conditional Bernoulli predictor (See Section IV) to predict the type of loss event, i.e., congestion loss or link corruption loss, and make TCP congestion window to behave accordingly. This is similar to speculation techniques that computer architecture community has used to eliminate the potential clock-cycle penalties caused by branch hazards. However, unlike the speculation techniques used in processor design, if the speculations are incorrect, there are no ways to "undo" or "flush" execution results in the case of TCP congestion control. This is because execution results of instructions in a processor are values of pre-designed calculations, while

TABLE I  
SIMILAR ISSUES EXIST IN BOTH PROCESSOR DESIGN AND TCP PROTOCOL DESIGN.

	Processor Design	Network Protocol Design
CTP (Critical to performance)	Execution time	Effective throughput
Problem	Control hazards degrade processor performance	Coexistence of two types of losses degrade TCP performance in wireless networks
What degrades performance	Two possible branching results	Two possible types of losses
Why degrades performance	Wasting time waiting for branching results	Wasting bandwidth responding to link corruption losses
A solution	Speculations (execute instructions as if branch predictions were always right)	Speculations (apply TCP congestion control as if all losses were due to link corruption)
Key behind speculation	Out-of-order execution; in-order commitment	Out-of-order loss differentiation; in-order packet retransmission
What if speculation was wrong	Undo or flush previous execution results	Improve speculation accuracy by minimizing congestion losses

execution results of the TCP congestion control algorithm are changes (increments or decrements) of the TCP sending speed.

An alternative approach to minimizing the impact of incorrect speculations is to improve the accuracy of the speculation. We propose to improve the speculation accuracy by minimizing the probability of congestion losses. This is done by optimally dimensioning the buffer of Explicit Congestion Notification (ECN) capable Random Early Detection (RED) gateways in the network. ECN [7] has been proposed by the Internet Engineering Task Force (IETF) to explicitly inform TCP senders of congestion at gateways, without requiring them to wait for either a retransmission timer timeout or three duplicate ACKs. ECN has been recommended to be used in conjunction with RED [8] [9].

If a RED buffer is optimally dimensioned with the thresholds appropriately set, the probability of congestion losses can be minimized by appropriately adjusting the sender's congestion window size based on feedback from ECN signals. Some preliminary work in minimizing packet losses at gateways have been reported in the literature [10] [11] [12]. The first study by Liu et.al. [10], instead of using a linear drop function and two thresholds as in RED, used only one threshold to mark packets; a packet is marked as congestion experienced (CE) with the probability of one if the average queue level exceeds the threshold. The study therefore, does not apply to gateways using RED. The second study by Kunniyur et.al. [11] did not study the effect of maximum threshold on packet drops at a RED gateway. Bai et.al. reported their results of a detailed investigation on this issue in their recent paper [12].

## III. THE PROPOSED SYSTEM ARCHITECTURE

Having described our motivations and the thought process, in this section, we present a system architecture that imple-

ments the idea we described previously. The discussions in this section are focused on the system level. More detailed description, analysis, and evaluation of each individual functional component of the system will be presented in the rest of this paper.

#### A. Design Challenges and Considerations

Our goal is to develop a system with protocol enhancements and/or additions to the existing TCP/IP protocol suite, to improve the network performance, in particular, the throughput, for networks with lossy links. A *lossy link* is defined as a communication link with high (e.g.,  $> 10^{-5}$ ) bit error rate. This includes RF wireless links, IR wireless links, satellite links, and even copper and fiber links exposed to high-dose rate radiation events. The wide definition is intended to target at many applications involving TCP/IP networks. However, in this paper, we will use RF wireless links (including RF satellite links) as an example to discuss and evaluate our algorithms. We do not intend to solve other significant issues for wireless networks such as handoff due to mobility, unless mobility can cause link errors (in most cases, it does). Given the background information and our intended approach, the following general requirements shall be considered for the system design.

1) *Effectiveness*: The system shall be able to effectively improve the network throughput over lossy links. The major issue caused by lossy links is that the existing TCP congestion control algorithm incorrectly interprets packet losses due to link errors as congestion losses, and correspondingly decreases the sender's congestion window size (i.e., transmission speed). This degrades the network throughput, and furthermore degrades the goodput. In order to effectively improve the network throughput over lossy links, algorithms are to be developed to either differentiate two types of losses, or quickly recover from congestion window size reductions.

2) *Ability to handle incorrect speculations*: Schemes shall be developed to handle incorrect speculations. Alternatively, speculation accuracy shall be guaranteed (if without schemes to detect incorrect speculations).

3) *Fairness*: The system shall be designed to be fair with competing flows. The window size based TCP congestion control was introduced to solve the fairness issue for each connecting flow. Whenever an enhancement is proposed, it should not break this law. This means end users equipped with new algorithms should contend for bottleneck bandwidth with other users who may or may not have enhancements in a fair way. To measure whether or not we meet this goal, a scientific metric should be defined (e.g., fairness index).

4) *Backward compatibility*: The new system shall be backward compatible with existing legacy networks, so that it will be easy to integrate such a new system with legacy networks. Since TCP protocol is an end-to-end performance regulation, theoretically all end users' TCP software should be modified to take the advantage of protocol enhancements. However this would not be practical in the real world. In the case of allowing system updates with protocol enhancements, serious consideration should be given to the system design, so that the

users with enhancements would be able to perform with the users with legacy protocols and perform fairly.

5) *Implementation complexity*: The system shall be easy to implement. Changes of and/or additions of protocols should be minimized. This decides the implementation complexity and in turn the overall costs. The general rule is we should leverage existing protocols and use existing networking devices as much as possible, and should avoid significant changes to the overall network system architecture.

#### B. System Functional Blocks and The Operation Process

With all the considerations in mind, we developed a cross-layer protocol-speculation based system architecture to address the challenges. The system functional blocks are shown in Figure 1. The system consists of protocol enhancements at three layers: middleware, transport, and network, among which, enhancements at the middleware layer and the network layer provide control functions and supporting parameters to transport layer functions. At the transport layer, we added a speculation function called conditional Bernoulli predictor (See section IV for details) providing inputs to a speculation based loss recovery and congestion window adjustment algorithms (called *SpecTCP*). The conditional Bernoulli predictor runs in parallel to the existing loss detection and loss treatment algorithms, both of which are selectively controlled by the congestion control algorithm manager at middleware layer.

At the network layer, we added a condition engine providing required information for the conditional Bernoulli predictor. The condition engine consists of mathematical models at ECN-capable RED gateways, producing networking parameters to maximize the prediction accuracy for the conditional Bernoulli predictor at the transport layer. The mathematical models define ECN queueing behaviors at RED gateways, and minimize the congestion losses at RED gateways. The ECN algorithm is mainly implemented at the network layer, but it provides early congestion indications to the speculative congestion control algorithm we added at the transport layer.

At the middleware layer, a simple software called congestion control algorithm manager is created to select and control the execution of congestion control schemes at the transport layer. Based on the global knowledge of a network, the manager makes and executes the decision whether the end system should run the legacy congestion control algorithm or the newer speculative congestion control algorithm.

Such end systems and gateways with cross-layer enhancements, if implemented in a network environment, operates collaboratively in a way illustrated in Figure 1. The general order of software execution is illustrated by arabic numbers in the figure. Loss events are detected by either retransmission timer timing out or three duplicated acknowledgements at the transport layer. Once a loss event is detected, the congestion control algorithm manager at the middleware layer will be informed and requested for commands. Using the global knowledge of the network, such as whether or not the receiver is ECN compatible, the congestion control algorithm manager will issue commands about which loss treatment algorithm

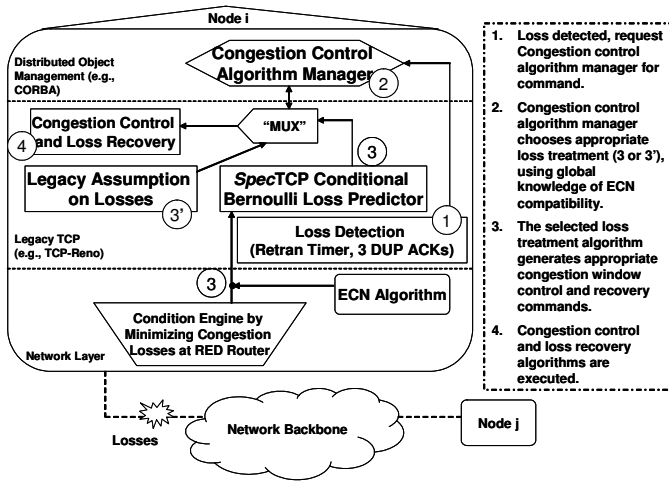


Fig. 1. The system-level operation process.

should be executed. If the receiver and other nodes along the way to receiver are ECN capable, then the speculative algorithm is applied. Otherwise the legacy algorithm is executed. Finally, the congestion window size is appropriately adjusted and lost packets are retransmitted.

### C. Discussions on The System Design

The proposed system architecture is effective in improving network throughput over lossy links. The proposed system is able to issue correct commands to the congestion control algorithm for different types of loss events. By speculations, the message sender does not have to waste time and bandwidth (congestion window size backoff) waiting for implicit network information about the losses. It solves the problem that the current TCP incorrectly interprets link corruption losses as network congestion losses. Therefore, it improves the network throughput effectively.

The proposed system is able to handle the issue of incorrect speculations. At the network layer, by minimizing network congestion losses, the speculation accuracy is maximized. In addition, for the small possibility of incorrect speculation, ECN algorithm is used to provide early explicit congestion signals. Therefore, when the speculation algorithm itself tends to incorrectly speculate a congestion loss (the fact) as a link corruption loss (the speculation result), the actual network congestion will be informed and handled by ECN algorithm.

The proposed system does not starve other competing flows. Under the normal working condition, no matter which congestion control algorithm is applied, all users are controlled by congestion window evolutions which was designed to reduce unfairness. The unfairness is highly likely to happen during failure modes of the system. For example, if the system incorrectly speculated a congestion loss as a link corruption loss, and ECN packets used to indicate congestions were lost, the system would not decrease the sender's congestion window size (but it should), which would result in starvation of other competing legacy TCP flows. To improve the ECN's reliability,

ECN packets are transmitted continuously until the sender acknowledges that ECN packets are received.

The new system is backward compatible with legacy networks. The congestion control algorithm manager at the middleware layer functions as a software bridge to ensure that end users with the new system can communicate with users with legacy networks. This is achieved by the ability of switching between the legacy congestion control algorithm and the propose speculative algorithm, based on the global ECN compatibility information obtained through middleware layer.

In order to implement such as system, the only new software is the congestion control algorithm manager at the middleware layer. However, given the function of this software, it should be fairly easy to implement and deploy. In addition, protocol modifications are needed at the transport layer by adding the speculation algorithm and interfaces with middleware and network layers. Some of these interfaces are there already, e.g., the interface between transport layer and network layer for ECN algorithm. At the network layer, almost nothing needs to be changed, except that the control parameters for ECN-capable RED gateways should be set using values suggested by our mathematical models for congestion loss minimization. This is a network management task, and may be implemented in the management software.

## IV. PROTOCOL-LEVEL SPECULATIONS

Motivated by the similar issues and solutions in processor design, in this section, we describe how we apply speculative techniques to the protocol design in details. We first describe conditional Bernoulli predictor, used to predict the type of loss events. The prediction function is implemented at the transport layer, but needs control and inputs from functions at middleware and network layers. We then present how we use prediction results for *SpecTCP*.

### A. Conditional Bernoulli Predictor

In this section, we describe the conditional Bernoulli predictor in details, and introduce some existing network predictors. We will compare the performance of conditional Bernoulli predictor with them in section VI.

1) *Conditional Bernoulli predictor architecture*: Most of hardware prediction algorithms used by computer architects are designed by a pure statistical state-machine-based approach. This means the transition from current prediction result to the next is controlled by a state machine where each state has probabilities of staying at the same state and transition to other states. These probabilities are calculated by a certain statistical distribution function which considers the most recent prediction result, i.e., taken or not taken. Unlike the 2-bit predictors for processor design, in a network, it is difficult to obtain the information whether or not the recent prediction result is taken. This has driven our design from a pure statistical state-machine-based prediction algorithm to a conditional probability prediction algorithm, in which

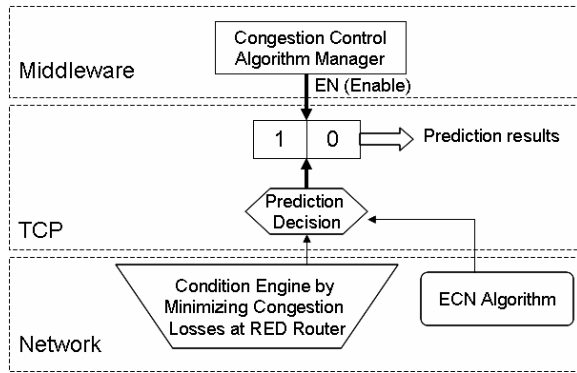


Fig. 2. The conceptual architecture of a conditional Bernoulli predictor.

we implement an engine at the network layer to feed the conditions needed by the prediction algorithm.

Figure 2 shows the conceptual architecture of a conditional Bernoulli predictor. The condition engine at the network layer uses our congestion loss minimization models (See section V for details) at RED gateways to minimize network congestion losses. The output of condition engine is that congestion loss events are minimized.

Based on the accuracy of this result, the prediction engine at the transport layer produces prediction results by setting the Bernoulli probability  $P$  for predicting link corruption losses, and accordingly the probability of predicting congestion losses is  $1 - P$ . If the condition engine is optimized, i.e., congestion losses are minimized, the prediction engine at the transport layer sets  $P = 1$ , which means predicting all incoming loss events as link corruption losses.

The congestion control manager at the middleware layer is responsible for enabling or disabling the conditional Bernoulli predictor. For network backward compatibility issues, i.e., some network components may not be ECN capable, the control manager may choose to disable the prediction function and switch back to the legacy congestion control algorithms. This function is especially useful for integrating new networking equipments with the existing network infrastructure.

In summary, the conditional Bernoulli predictor leverages the design concept of 2-bit hardware predictor for processor design. However, its probability of staying at the same state, as well as the probability of transition, are controlled and pre-determined by the condition engine at the network layer. In addition, the overall architecture is implemented in software across three network layers.

2) *Other network predictors in the literature*: The purpose of reviewing other existing predictors in this section is to give an introduction to some previous work, with which we can compare the performance of the conditional Bernoulli predictor. There are many hardware predictors proposed for computer architecture. As mentioned previously, there are some differences between these predictors and the proposed conditional Bernoulli predictor. It is therefore technically inefficient to implement some most advanced computer predictors (more complex than a 2-bit predictor) in a wireless networking

	Conditional Bernoulli Predictor	Vegas-sender	NTG-sender	NDG-sender	Predictor-receiver
Where to implement	sender	sender	sender	sender	receiver
Minimum NO. of history to track (amount of memory)	none	1	2	2	3
Other network performance parameters required	none	RTT, Congestion window size	RTT, Congestion window size	RTT, Congestion window size	Packet inter-arrival time
Help from other layers? (Y/N)	Yes (need zero loss control at network layer - RED router)	No	No	No	No
Implementation complexity	Medium (cross layer - require control at router)	High (require computation of multiple network parameters)	High	High	Low (only need to monitor inter-arrival time)
Compatibility with legacy TCP/IP	Require RED router	compatible	compatible	compatible	compatible

Fig. 3. Qualitative comparison of conditional Bernoulli predictor with Vegas-Sender, NTG-Sender, NDG-Sender, and Predictor-Receiver

environment.

However, it is reasonable to compare the conditional Bernoulli predictor with some existing prediction algorithms designed for networks, as well as its design prototype, 2-bit hardware predictor for computers. Before we show the detailed performance comparison results in section VI, we would like to briefly describe some of predictions algorithms designed for networks in the literature. Authors in [13] and [14] proposed four types of predictors used to improve the network throughput over wireless links. They include one predictor at the receiver and three types of predictors at the sender. We qualitatively compare our conditional Bernoulli predictor with the four types predictors proposed in [13] and [14], and the qualitative comparison results are presented in Figure 3.

### B. SpecTCP Algorithm

Figure 4 shows the kernel of our proposed *SpecTCP* algorithm at the sender's side. As the Bernoulli probability is set to  $P = 1$ , a *SpecTCP* sender treats retransmission timer timing out and/or three duplicate acknowledgements as indications of link errors. In this case, the *SpecTCP* source does not decrease *cwnd*, even though the speculation correctness is unknown. As shown in Figure 4, this speculation is based on the condition that no ECN\_ECHO packets are received (i.e., potentially no congestion losses). Therefore, in order to improve the speculation accuracy, a scheme is required to minimize losses due to network congestion. We will discuss it in section V. If incorrect speculation was made, i.e., the *SpecTCP* sender receives the ECN\_ECHO packet sent by the receiver, the sender treats it as network congestion and triggers the Fast Recovery algorithm [15] as in the current TCP.

In *SpecTCP*, the congestion window size is appropriately controlled in the presence of either network congestion or corruption in the following way. Congestion window is halved using Fast Recovery algorithm when there is network congestion (explicitly notified by ECN\_ECHO packets), and persists

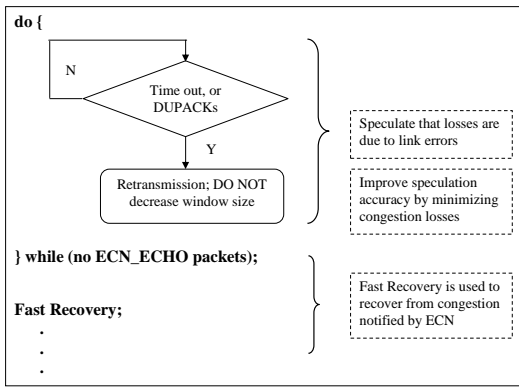


Fig. 4. *SpecTCP* software kernel.

at the previous value in the presence of corruption. There are two mechanisms that might be applied to adjust congestion window when *SpecTCP* sender detects corruption: (i) keep *cwnd* unchanged as the previous value; (ii) use Congestion Avoidance algorithm to slowly increase *cwnd*. In our algorithm, we adopt the first mechanism: make congestion window persist in the previous value.

ECN mechanism will be most effective if it is used with active queue management (such as RED) [15]. In active queue management, when a buffer reaches a certain threshold, the gateway will send a CE packet to the TCP receiver. Gateways send CE packets before their buffers overflow. Therefore, packet drops due to congestion happen only after the gateway has sent CE packets.

Upon receiving the CE packet, the TCP receiver will keep sending ECN\_ECHO packet back to the sender until it receives a CWR packet from the sender, which means the sender has responded to network congestion. The sender only responds to the first ECN\_ECHO packet and ignores others up to one RTT.

Depending on the threshold of RED and the level of network congestion, ECN\_ECHO packets can arrive at the sender either before or after the retransmit timer times out due to congestion packet losses (caused by buffer overflow). Our proposed *SpecTCP* is effective in both cases as described below.

- Case 1: Timer times out after ECN\_ECHO packets are received by the sender.

In this case, the sender will respond to congestion as indicated by the receipt of ECN\_ECHO packets. This case is desirable.

- Case 2: Timer times out before ECN\_ECHO packets received by the sender.

In this case, the retransmit timer timeout happens at time  $t_1$ , and ECN\_ECHO packets are received by the sender at time  $t_2$  ( $t_1 < t_2$ ). If the difference between  $t_1$  and  $t_2$  is small enough, though the TCP sender does not respond to packet losses indicated by retransmit timer timeout at  $t_1$ , ECN\_ECHO packets will arrive very quickly, which will

trigger Fast Recovery mechanism to relieve the network out of congestion.

The difference between  $t_1$  and  $t_2$  can be decreased by decreasing  $t_2$  as described below. As mentioned above, using active queue management such as RED, when buffer reaches threshold, it will send the CE packet to receiver. Upon receiving the CE packet, the receiver starts to send ECN\_ECHO packets to the sender. Though it is difficult to control the travel time of ECN\_ECHO packets from the receiver to the sender, we can make the receiver send ECN\_ECHO packets earlier by letting the gateway send the CE packet earlier. The earlier ECN\_ECHO packets are sent, the earlier they arrive at the sender, i.e., the smaller the value of  $t_2$  is. The time when the gateway sends the CE packet is decided by the value of threshold. Therefore, an optimum value of RED's threshold is very important for the sender to receive congestion notification quickly. Optimal RED threshold is one of our current research topics.

## V. IMPROVING SPECULATION ACCURACY: MINIMIZING CONGESTION LOSSES

In this section, we develop a model to analyze the performance of ECN mechanism in RED gateways, and furthermore, we derive the expressions for the maximum buffer size and the maximum threshold of a RED gateway to minimize congestion packet losses. The minimization of congestion losses significantly improves the accuracy of speculating that loss events are due to link corruptions. This indicates that the condition engine within conditional Bernoulli predictor is optimized. Therefore, it is reasonable to set  $P = 1$ , i.e., to predict all incoming loss events are caused by link errors.

ECN-capable RED gateways use an exponential weighted moving average to calculate an average queue size from the instantaneous queue size, and two thresholds (*minimum* and *maximum*), to determine whether an arriving packet should be dropped. If the average queue size is greater than the maximum threshold, the packet is *dropped*. If the average queue size is between the minimum and the maximum thresholds, the packet is marked with a probability as a Congestion Experienced (CE) packet.

Packet losses due to the average queue size exceeding the maximum threshold at a RED gateway degrade TCP performance. The *objective* of this section is to determine if packet losses at a gateway can be minimized by optimally dimensioning the buffer and selecting the two RED thresholds.

### A. Notations

We consider a model consisting of two RED gateways fed by multiple sources. The link connecting two RED gateways is the bottleneck link which causes congestion. The sources, destinations and the RED gateways use ECN for end-to-end congestion control. The following notations will be used in our model:

- $Q(t), Q(t)_{max}$ : *Instantaneous* and *maximum instantaneous* queue sizes respectively at the RED gateway at time  $t$ .

- $\bar{Q}$ ,  $\bar{Q}_{max}$ : Average and maximum average queue sizes respectively at the RED gateway.
- $\omega$ : Weighting factor for calculating  $\bar{Q}$ .
- $p(t)$ : Marking probability at the RED gateway at time  $t$ .
- $min_{th}$ ,  $max_{th}$ : Minimum and maximum thresholds respectively of a RED gateway.
- $m$ : total number of TCP flows.
- $W_i(t)$ : Window size of the  $i^{th}$  TCP flow at time  $t$ ,  $t \geq 0$ ,  $i = 1, \dots, m$ .
- $SStresh_i$ : Slow Start threshold for the  $i^{th}$  TCP flow,  $i = 1, \dots, m$ .
- $r_i$ : Round Trip Time (RTT) for the  $i^{th}$  TCP flow,  $i = 1, \dots, m$ .  $r_i$  is replaced by  $r$  when all the RTTs are same.
- $\bar{\mu}_i$ : Average share of bottleneck link bandwidth of the  $i^{th}$  TCP flow,  $i = 1, \dots, m$ .
- $\mu$ : Bandwidth of bottleneck link which is given by  $\mu = \sum_{i=1}^m \bar{\mu}_i$ .
- $T[1]$ : Waiting time for the first marking event after the average queue size exceeds  $min_{th}$  [10].
- $\beta_i$ : Number of window size increases during time  $T[1]$  for the  $i^{th}$  TCP flow,  $i = 1, \dots, m$ .
- $\tau_i$ : Propagation delay from source  $i$  to the RED gateway,  $i = 1, \dots, m$ .
- $t_0$ : Time when the first packet is marked at the RED gateway [10].
- $t_1$ : Time when the last packet, which was sent just before the first window size reduction, arrives at the RED gateway [10].

For every packet arrival, the RED gateway estimates  $\bar{Q}$  using the following exponential weighted moving average algorithm

$$\bar{Q} \leftarrow (1 - \omega)\bar{Q} + Q(t)\omega, \quad (1)$$

and then calculates the packet marking/dropping probability  $p(t)$  using

$$p(t) = \begin{cases} 0, & 0 \leq \bar{Q} < min_{th}; \\ \frac{\bar{Q} - min_{th}}{max_{th} - min_{th}} max_p, & min_{th} \leq \bar{Q} \leq max_{th}; \\ 1, & \bar{Q} > max_{th}. \end{cases} \quad (2)$$

### B. Assumptions

We make the following assumptions regarding RED gateways and TCP sources in our analytical model for minimizing packet losses in Secs. V-C and V-D.

- For small  $\omega$  (as suggested in [8]),  $\bar{Q}$  varies very slowly, so that consecutive packets are likely to experience the same marking probability [16].
- The random packet marking of packets in flow  $i$  is described by a Poisson process with time varying rate  $\lambda_i(t) = p(t)W_i(t)/r_i(t)$  [17]. Accordingly, the waiting time ( $T_i[n]$ ) for the  $n$ -th marking event of flow  $i$ , which is given by  $T_i[n] = \sum_{k=1}^n X_i(k)$ , is a Gamma distributed random variable.  $X_i(k)$  is the time interval between  $(k-1)$  and  $k$ -th marking events for flow  $i$ . Specifically, the expected value of the waiting time for the first marking event is  $E[T_i[1]] = 1/\lambda_i(t)$ .

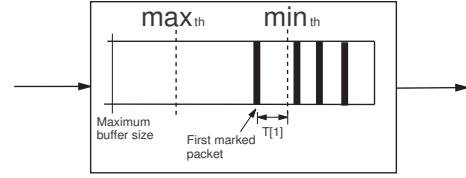


Fig. 5. Analytical model of a RED gateway.

- All TCP sources start sending at the same time, and all packets are of the same size (as used in [10]). The queue size is measured in packets.

### C. Maximum Buffer Size

Packet drops at an ECN-capable RED gateway are either due to buffer overflows ( $Q(t)$  is equal to the buffer size) or  $\bar{Q} > max_{th}$ . In this section, we estimate the buffer size required for minimizing packet losses.

The congestion window size during the slow start phase increases very quickly. The average queue size (being the output of a low pass filter) of a RED gateway can not follow the quick change of  $Q(t)$ ; as a result  $\bar{Q}$  stays less than  $min_{th}$ . Therefore,  $Q(t)$  reaches the maximum value when the packet leaving the source at  $t - \tau_i$  reaches the RED buffer. When this packet left the source,  $W_i(t - \tau_i) = SStresh_i$  for  $i = 1, 2, \dots, m$ ; the queue size is smaller when the sources are in congestion avoidance [10]. For  $m$  TCP flows,  $Q(t)_{max}$  can be expressed as the output of a system with processing capacity of  $\sum_{i=1}^m r_i \bar{\mu}_i$  and the maximum input rate when sources reach their slow start threshold.

$$Q(t)_{max} = \sum_{i=1}^m (W_i(t - \tau_i) - r_i \bar{\mu}_i) = \sum_{i=1}^m (SStresh_i - r_i \bar{\mu}_i). \quad (3)$$

$Q(t)_{max}$ , as given by the above equation, is therefore, the **buffer size required to minimize packet loss at the RED gateway**.

### D. $max_{th}$ for RED gateways

Authors in [8] have recommended  $max_{th} = 3 \times min_{th}$ . In this section, we setup a model to estimate  $max_{th}$  for minimizing losses at the RED buffer. We start with the recommended RED parameter values, and end with values suggested by our model.

Figure 5 shows our analytical model of a RED gateway. When the average queue size is in the steady-state condition (during which the sources are in the congestion avoidance phase), the instantaneous queue size at time  $t_0$  is

$$Q(t_0) = min_{th} + \sum_{i=1}^m \beta_i, \quad (4)$$

where  $\beta_i$  can be calculated as

$$\beta_i = \frac{E[T[1]]}{r_i} = \frac{1}{\lambda_i(t)r_i} = \frac{1}{p(t)W_i(t)}, i = 1, \dots, m. \quad (5)$$

Since the difference between  $t_0$  and  $t_1$  is one RTT, and the window size of a source is increased by one per RTT during

the congestion avoidance phase, the instantaneous queue size at time  $t_1$  can be expressed as

$$Q(t_1) = \min_{th} + \sum_{i=1}^m (\beta_i + 1). \quad (6)$$

The average queue size is estimated using an exponential weighted moving average as shown in Equation (1). If time is discretized into time slots with each slot being equal to one RTT, the RED's average queue size estimation algorithm at the  $k$ -th slot can be expressed as

$$\bar{Q}[k+1] = (1-\omega)\bar{Q}[k] + Q[k]\omega. \quad (7)$$

In practice,  $\omega$  is very small, and the congestion window size increases by one every RTT during the congestion avoidance phase. Therefore, before the first marking event happens (i.e., no congestion control) it is reasonable to consider both the instantaneous queue size and the average queue size to be constant within a very short time period (see the first assumption in Section V-B). Thus, by plugging  $Q(t_1)$  (slot  $k$  is equal to  $t_1$  in time) into Equation (7) and assuming that the average queue sizes during the two previous consecutive time slots are the same, the average queue size estimated at time  $t_1$  can be solved iteratively, which is

$$\bar{Q}_{max} = \bar{Q} = \min_{th} + \sum_{i=1}^m (\beta_i + \omega). \quad (8)$$

The first marking event is followed by many random ECN marking events, which make TCP sources adjust their congestion window sizes. The average queue size stays at a certain level smaller than the average queue size at time  $t_1$ , as will be shown by our simulation results in section VI later. Therefore, **Equation (8) gives the maximum average queue size for minimizing packet losses, i.e. this is our suggested value of  $max_{th}$ .**

### E. Calculating the Average Share of Bottleneck Link

Equation 8 suggests the maximum threshold for a RED gateway to minimize congestion losses. The parameter  $\beta_i$  is calculated by Equation 5, but it is not straightforward to measure and configure. However, the parameter  $m$ , the total number of TCP flows, needs to be bounded. Otherwise,  $\bar{Q}_{max}$  is not the maximum value mathematically.

Assuming  $\alpha$  is an integer, then  $m$  should satisfy the following inequality.

$$1 \leq m \leq \alpha, \quad (9)$$

where  $\alpha$  is calculated by Equation 10:

$$\alpha = \frac{BW_{bn}}{\bar{\mu}_i}. \quad (10)$$

The  $BW_{bn}$  is the bandwidth of bottleneck link in the network; and the  $\bar{\mu}_i$  is the average share of the bottleneck link bandwidth of the  $i$ th flow. In Equation 10,  $BW_{bn}$  is known when a network is set up. In this section, we focus on the

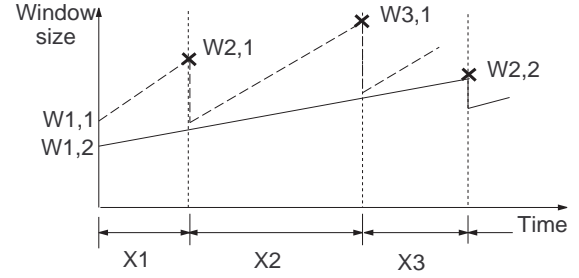


Fig. 6. The window evolution approximation with two TCP-Reno flows.

calculation of the average share of bottleneck link  $\bar{\mu}_i$ , and in turn, we will get the bounded value for  $m$ .

We consider a model consisting of a bottleneck link fed by  $i$  TCP-Reno sources. Figure 6 shows the window evolution approximation for TCP-Reno sessions with different round trip delays sharing a bottleneck link with a RED gateway. The loss events are represented by 'x' marks. This approximation has been used by many researchers [18], [19] for the analytic understanding of the RED performance. Based on Figure 6, we use the following additional notations in our modeling:

$w_{i,j}(t)$  = The  $j$ th TCP session's window size right before the previous loss event.

$w_{i+1,j}(t)$  = The  $j$ th TCP session's window size right before the current loss event.

$w_{javg}(t)$  = The time-average window size for the  $j$ th TCP session.

$r_{i,j}$  = Round Trip Time (RTT) of the  $j$ th TCP session.

$\bar{\mu}_i$  = Average share of the bottleneck link bandwidth.

$L_i$  = The time when  $i$ th congestion loss event happens.

Consider the scenario in Figure 6;  $X_i = L_i - L_{i-1}$  denotes the inter-loss duration. The window evolution could be expressed by the following equation.

$$w_{i+1,j}(t) = \frac{w_{i,j}(t)}{2} + \frac{X_i}{r_{i,j}}. \quad (11)$$

Since loss events are determined by both the traffic type and the random marking at RED gateway, it is reasonable to consider  $\{X_i\}$  as an Independent Identical Distributed (i.i.d.) *renewal process*. If we assume in any length of time interval, the number of loss event is Poisson distributed, then the total number of loss events in the interval  $(0, t)$  is a *Poisson process*, denoted by  $N(t)$ . Therefore, the loss time interval  $X_i$  is an i.i.d. *exponential random variable* with the parameter  $\lambda$ . Its probability density function (pdf) is

$$f_{X_i}(t) = \lambda e^{-\lambda t} u(t). \quad (12)$$

In addition, the waiting time  $T[n] = \sum_{k=1}^n X_k$  for a loss event is a *gamma distributed random variable* with parameters  $(n, \lambda)$ . Its pdf can be found as

$$f_T(t) = \frac{\lambda e^{-\lambda t} (\lambda t)^{n-1}}{\Gamma(n)} u(t), \quad (13)$$

which is, in the other form,

$$f_T(t) = \frac{\lambda^n e^{-\lambda t} t^{n-1}}{(n-1)!} u(t). \quad (14)$$

Based on the mathematical nature of the window evolution we analyzed above, we finally calculate the average share of bottleneck link bandwidth, which has been defined as

$$\overline{\mu_i} = \frac{\overline{w_{j_{avg}}(t)}}{r_{i,j}}. \quad (15)$$

Taking the expectation for both sides of Equation (11), we have

$$\overline{w_{i+1,j}(t)} = \frac{\overline{w_{i,j}(t)}}{2} + \frac{\overline{X_i}}{r_{i,j}}. \quad (16)$$

Since any two loss events have the same statistical characteristics, it is apparent that  $w_{i+1,j}(t)$  and  $w_{i,j}(t)$  have the same expected value. Thus,

$$\overline{w_j(t)} = 2 \frac{\overline{X_i}}{r_{i,j}}. \quad (17)$$

Recall that the loss time duration is a renewal process and the total number of loss events during any length of time interval is a Poisson process, from Equation (12), we should have

$$\overline{X_i} = E[X_i] = \frac{1}{\lambda}. \quad (18)$$

Therefore,

$$\overline{w_j(t)} = \frac{2}{\lambda r_{i,j}}. \quad (19)$$

Because Poisson process is ergodic in mean, using the property of ergodicity, we have

$$\overline{w_{j_{avg}}(t)} = \overline{w_j(t)} = \frac{2}{\lambda r_{i,j}}. \quad (20)$$

Finally, the average share of the bottleneck link bandwidth is

$$\overline{\mu_i} = \frac{\overline{w_{j_{avg}}(t)}}{r_{i,j}} = \frac{2}{\lambda r_{i,j}^2}. \quad (21)$$

Remarkably, the above result implies that the connection with the shortest RTT has the largest average share, which is also found by authors in [10], [20]. Equation 21 shows that  $\overline{\mu_i}$  is a function of the measurable and configurable  $r_{i,j}$  (given  $\lambda$  is a constant for Poisson process). Therefore,  $\alpha$  in Equation 9 is found and  $m$  is bounded.

## VI. PERFORMANCE EVALUATIONS

In this section, we present system performance evaluation results using simulations. We implement the congestion control algorithm manager at the middleware layer, the conditional Bernoulli predictor and the SpecTCP at the transport layer, the congestion loss minimization models (i.e., optimizations used to improve the speculation accuracy) at the network layer, and the required software interfaces. In the following sections, we will first present our simulation configurations and evaluation results for the congestions minimization models

TABLE II  
COMPARISON BETWEEN SIMULATIONS AND ANALYTICAL MODELS FOR  
MAXIMUM BUFFER SIZE (PACKETS).

Sim cases	$SSt_{h1}$	$SSt_{h2}$	$r = r_1 = r_2$ (ms)	$\mu$ (Mbps)	$Q(t)_{max}$	
					Anlyt	Sim
Case 1	15	15	59	1.5	<b>19</b>	<b>18</b>
Case 2	15	20	59	1.5	<b>24</b>	<b>23</b>
Case 3	15	15	99	1.5	<b>12</b>	<b>13</b>

TABLE III  
COMPARISON BETWEEN SIMULATIONS AND ANALYTICAL MODELS FOR  
MAXIMUM THRESHOLD (I.E., MAXIMUM AVERAGE QUEUE SIZE).

Sim cases	$\omega$	$min_{th}$ (Pkts)	$max_{th}$ (Pkts)	$max_p$	$\overline{Q}_{max}$ (Pkts)	
					Anlyt	Sim
Case 1	0.002	5	15	0.1	<b>7.3</b>	<b>7.6</b>
Case 2	0.002	5	15	0.2	<b>6.7</b>	<b>7.1</b>
Case 3	0.002	7	21	0.1	<b>9.6</b>	<b>9.9</b>

(i.e., speculation accuracy models). We then describe simulation configurations and evaluation results in a network with competing flows for the overall system. Evaluation results are presented in two steps: first, we compare the performance of our proposed scheme with the baseline network, i.e., the legacy TCP/IP network with TCP-Reno; secondly, we compare our proposed scheme with Snoop [21] and TCPW [22].

### A. Evaluating Speculation Accuracy Improvement Models

We have evaluated the models using *ns-2*. We used three configurations as described below.

1) *Verification of the maximum buffer size*: To verify the maximum buffer size suggested by our model in Sec. V-C, we have run simulations for three different cases with different values of  $r$  and  $SSt_{hresh}$  as shown in Table II. We have measured  $Q(t)_{max}$ , and the results are compared with  $Q(t)_{max}$  predicted by our analytical models. It is seen that values from simulation and analytical models are close, thereby validating the maximum buffer size suggested by our analytical model.

2) *Verification of  $max_{th}$* : The RED parameters shown in Table III are used to verify the correctness of the value of  $max_{th}$  suggested by our model in Sec V-D. Case 1 uses recommended RED parameters. To make the different cases comparable, we choose RTT of all TCP connections to be the same (59 ms). As shown by the results, the  $\overline{Q}_{max}$  (which we have suggested in Equation (8) as the value to be used for  $max_{th}$ ) obtained from our analytical model agrees with the one obtained from simulation.

The congestion window size, instantaneous queue size, and average queue size for Case 1 in Table III are shown in Figure 7. We can see that the instantaneous queue size is maximum when the congestion window size reaches the slow start threshold. The average queue size is maximum just before the first marking event at time  $t = 1.9s$ . This proves the validity of our statement in Sec. V-C about the instantaneous queue size reaching the maximum. The same observation can

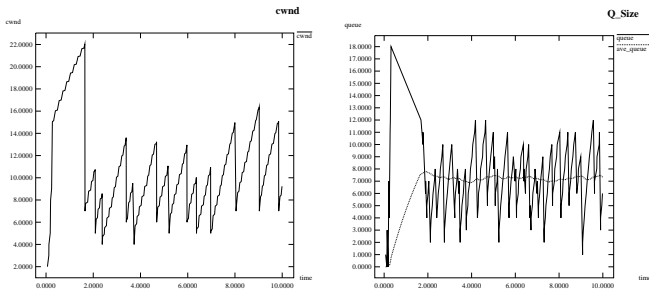


Fig. 7. Congestion window, instantaneous queue size and average queue size for Case 1.

TABLE IV  
SIMULATION RESULTS OF PACKET DROPS WITH DIFFERENT RED  
PARAMETERS.

Simulation cases	$\omega$	$min_{th}$ (Packets)	$max_{th}$ (Packets)	$max_p$	Pkt drops
Case 1	0.002	5	15	0.1	No
Case 2	0.002	5	8	0.1	No
Case 3	0.002	5	7	0.1	Yes

be made for the other cases; due to space limitations we do not present the results for the other cases.

3) *Verification of the optimality*: The third simulation shown in Table IV is used to verify the optimality our estimated  $max_{th}$ . Case 1 in Table IV uses recommended RED parameters. It works perfectly to control the TCP congestion without unnecessary packet drops. However, case 2 in Table IV uses our proposed  $max_{th}$ , which is much smaller than the recommended value. The simulation result shows it achieves the same congestion control effects as the recommended value does. In addition, one of the benefits of using our estimated value of  $max_{th}$  is that the queuing delay and buffer size can be significantly reduced (though it is not shown by simulation here). Case 3 is used to verify the validity of our proposed model. The value of  $max_{th}$  is reduced from 8 estimated using our proposed model to 7. The result of this reduction is that the objective of no packet drops can never be achieved.

### B. SpecTCP Performance Evaluation Against the Baseline

In this section, we present our simulation comparison of the proposed scheme with the baseline scheme. We first evaluate *SpecTCP* throughput by comparing it with TCP-Reno with ECN capability. We then present the frequency of mispredictions for *SpecTCP* in different BER environments. Finally, we present the congestion loss rate in different BER settings.

We have evaluated the performance of our *SpecTCP* algorithm using *ns-2*. The ECN implementation is based on RFC 2481 [15]. Two local area networks (10 Mbps) are connected by a 64 Kbps lossy link with a propagation delay of 280 ms. RED gateways are used in our simulations to set the CE bit in the packet header. The full-duplex link between gateway A and gateway B has a BER between  $1e^{-7}$  to  $1.2e^{-4}$  in our simulations. The receiver's advertised window size, which is

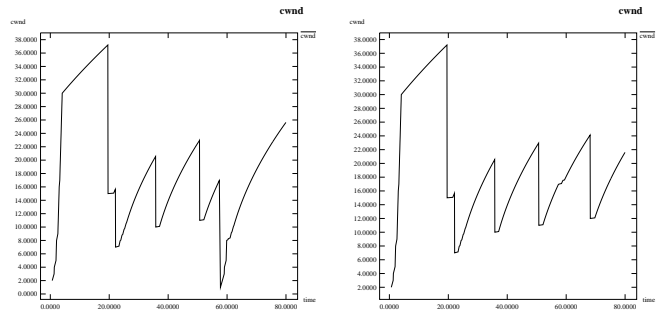


Fig. 8. Comparison of congestion window for TCP-Reno with ECN (left) and SpecTCP (right).

also equal to the initial  $ssthresh$  at the sender, is set to 30 segments.

1) *Comparing congestion window evolutions*: In this evaluation, we created multiple packet losses due to corruption at time 57 sec, and network congestion losses at 19 s, 22 s, 34 s and 50 s.

Comparing window evolutions in Figure 8, we find that, when SpecTCP is used, the congestion window never goes down in the presence of packet losses due to link errors; instead, it persists in the previous value as expected. The Fast Recovery mechanism is triggered in the presence of network congestion (notified by ECN\_ECHO packets). However, when the current TCP (with ECN capability) is used in the simulation, the timeout caused by packet losses triggers the Slow-Start mechanism that results in the reduction of congestion window to the initial value. The reason is that the current TCP (though it is ECN capable in our simulation) makes the assumption that all losses are caused by congestion. Thus, when there is packet loss, no matter whether it is caused by congestion or corruption, the current TCP with ECN capability triggers TCP's congestion control mechanism. Since SpecTCP predicts the causes of packet errors (prediction accuracy is guaranteed at network layer), and network congestion is explicitly notified by ECN\_ECHO packets; TCP's congestion control mechanism is not triggered when packets are lost due to link errors.

2) *Evaluating throughput*: Goodput (the amount of useful information, in *bit*, being received by the receiver per second, not including errors) obtained from simulation experiments for both *SpecTCP* and TCP-Reno with ECN capability are compared.

Figure 9 compares the goodput in bit/s and the normalized throughput of both *SpecTCP* and TCP-Reno with ECN capability. At the BER of  $5e^{-5}$ , the goodput of our *SpecTCP* is almost 5 times higher than that of TCP-Reno with ECN. From Figure 9, this improvement is much higher at higher BER. In addition, the throughput of TCP-Reno with ECN suffers more severely than our *SpecTCP* as the error rate increases. We can also see that, with the increase of BER, the throughput of TCP-Reno with ECN decreases much faster than our *SpecTCP*. For example, according to Figure 9, when BER increases from  $1e^{-5}$  to  $5e^{-5}$ , TCP-Reno's goodput

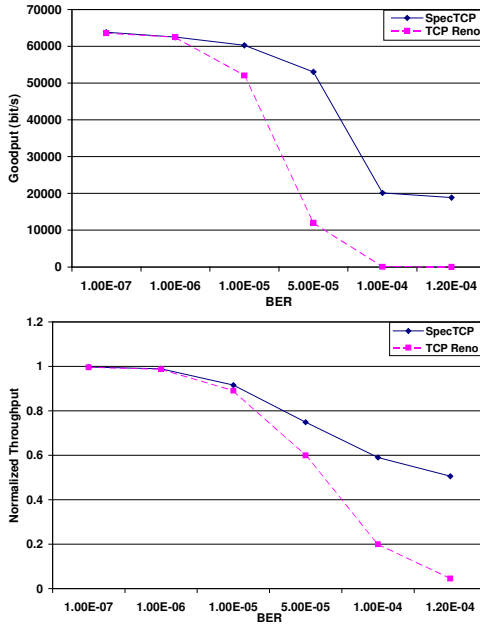


Fig. 9. Comparison of goodput (bit/s) (up) and normalized throughput (down).

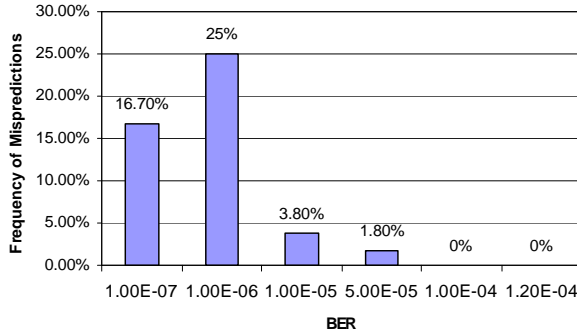


Fig. 10. Misprediction rate for SpecTCP.

decreases by **77 %** in contrast to our *SpecTCP* whose goodput only decreases by **12 %**. This is because, at higher BER, congestion window reductions for TCP-Reno is so frequent that congestion window size cannot reach a high value.

3) *Misprediction rate*: *SpecTCP* (configured by the conditional Bernoulli predictor) speculates that losses are due to link errors. Though we have shown the effectiveness of our model to improve the speculation accuracy, there must be mispredictions in reality. Misprediction rate is defined as the frequency when *SpecTCP* mispredicts a congestion loss as a loss due to link errors. The simulation results are shown in Figure 10. As shown, with our analytical model to guarantee the speculation accuracy, *SpecTCP* speculates accurately, especially in high BER settings. Misprediction rates are a little higher at smaller BER values (i.e.,  $1e^{-7}$  and  $1e^{-6}$ ), but they still reflect similar performances as those speculation algorithms used by computer architects [2].

4) *Congestion loss rate*: One of the benefits that ECN brings into networks is a network with ECN scheme has

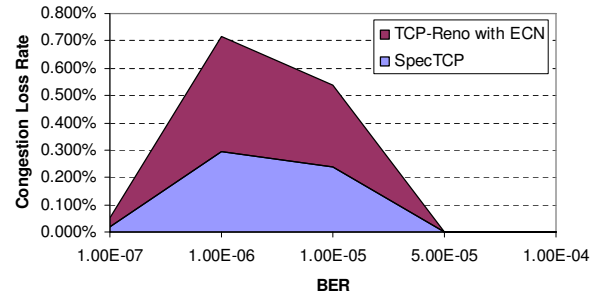


Fig. 11. Congestion loss rate for SpecTCP.

less congestions losses. Figure 11 shows that our proposed scheme has much lower congestion loss rate in all BER settings, comparing with the legacy TCP-Reno with ECN capability. This indicates that our congestion loss minimization models are effective in reducing network congestion losses. Furthermore, the congestion loss minimization effort itself in this paper is significant and useful.

It is observed that, when BER is small, the congestion loss rates for both schemes are low. Typical wired links have such BER values. In a wired network with ECN capability, end users are pre-notified of future network congestion events, thus adjusting TCP window size (i.e., user sending speed) accordingly to prevent severe network congestions. Similar results are observed when BER is large. These BER values are even larger than typical BER values for a RF wireless network. In this case, frequent link errors significantly disrupt the normal network transmissions. Therefore, end users' TCP congestion window sizes can hardly increase to the level causing network congestions.

5) *Fairness in a network with competing legacy TCP flows*: The impact of a *SpecTCP* connection on other TCP connections, especially, other legacy TCP (e.g, TCP-Reno) connections is evaluated. The purpose is to ensure the proposed scheme does not steal bandwidth from other legacy TCP flows, or in the worse case, starve other flows. The definition of fairness, and how to measure it, differs from case to case. However, most of researchers use the definition that Jain defined in [23]. It is shown in Equation 22, where  $x_i$  is the throughput for the  $i^{th}$  user.

$$FairnessIndex = \frac{(\sum_{i=1}^n x_i)^2}{n \sum_{i=1}^n x_i^2}. \quad (22)$$

Following this definition, we measured the fairness index between the proposed *SpecTCP* and the legacy TCP-Reno with ECN capability. Figure 12 shows evaluation results. As seen in Figure 12, when BER is small, *SpecTCP* flows performs fairly with TCP-Reno with ECN. When BER is large, *SpecTCP* performs more aggressive (with minimized congestion loss control at RED gateways) than TCP-Reno with ECN. It is little unfair to Reno but reasonable and acceptable.

### C. Performance Comparisons with Related Work

In this section, we first compare the performance of our proposed conditional Bernoulli predictor with other network

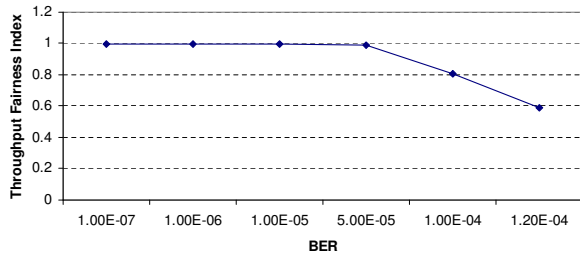


Fig. 12. Fairness in a network with TCP-Reno competing flows.

predictors in the literature. We then compared the performance of *SpecTCP* with a number of previous work. They include TCP with ECN, Snoop, and TCPW (TCP-Westwood). Identical simulations are setup for all these schemes, and results are presented below.

1) *Performance comparison for the conditional Bernoulli predictor*: Hardware branch predictor is an approach that computer architects use to improve instruction-level parallelism. There have been many branch prediction algorithm proposed to improve prediction accuracy with low overhead and hardware implementation complexity. Branch prediction by these algorithms is based on the information whether or not the recent prediction is taken. In networks where we try to predict whether or not a loss is due to congestion or link corruption, it is not cost-effective to obtain such information. We therefore compare the performance of our predictor with those (See section IV for details) proposed for similar purpose in network environments.

In addition to the qualitative comparison we performed in section IV, in this section, we measure the misprediction rates for all of them. Figure 13 shows the proposed conditional Bernoulli predictor has the best performance. In addition, it shows the proposed conditional Bernoulli predictor has the similar performance to a 4096-entry 2-bit predictor widely used in computer processor design. The concept of 2-bit predictor forms the conceptual basis of our proposed conditional Bernoulli predictor for network protocol design.

2) *Performance comparison for the network*: Identical simulations are configured for all TCP improvement schemes. Figure 14 compares the congestion loss rate (the ratio of congestion losses to the total number of transmitted packets) among *SpecTCP*, TCP with ECN, Snoop, and TCPW. *SpecTCP* has the smallest congestion loss rate. This is due to the deployment of ECN based congestion control and our congestion loss minimization model (used to guarantee speculation accuracy).

Figure 15 compares the normalized Goodput (Goodput in bps divided by the total number of transmitted packets) among four TCP improvement algorithms. As shown in Figure 15, *SpecTCP* outperforms other three algorithms. In addition, we observed that, when BER is small, the proposed *SpecTCP* has similar goodput to TCP-Reno with ECN capability. From Figure 14, at the same BER values, the congestion loss rates for both scenarios are also very low. Because they both have similar low BER (link errors) and congestion loss rate (errors

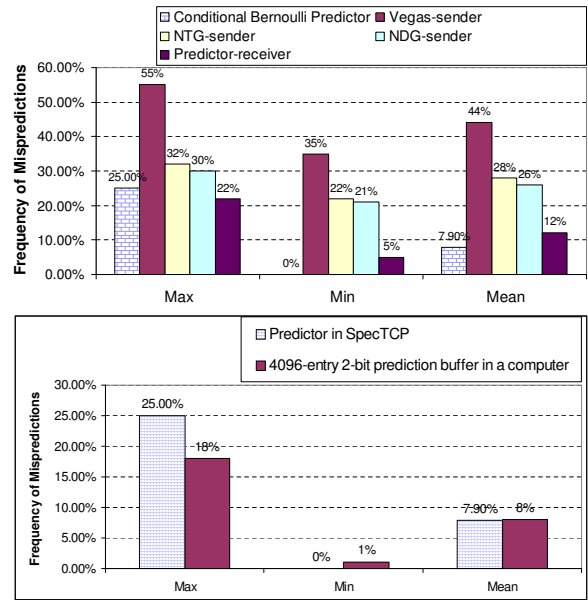


Fig. 13. Comparison with other network predictors (up), and a 4096-entry 2-bit predictor for computer (down).

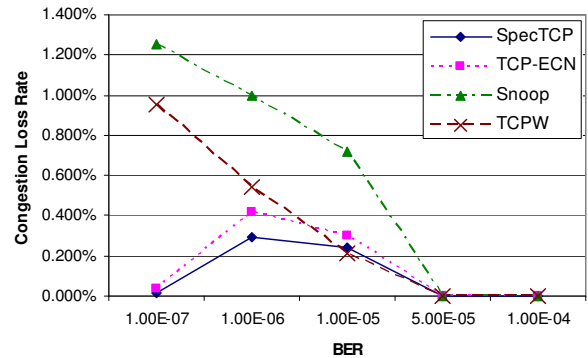


Fig. 14. Comparison of congestion loss rate.

due to congestions), both scenarios should have high goodput. For Snoop and TCPW, since they do not minimize congestion losses, they get smaller goodput.

## VII. CONCLUSIONS AND FUTURE WORK

We have developed a cross-layer speculative system architecture to improve throughput for computer networks over lossy links. The proposed system consists of protocol enhancements at middleware, transport, and network layers, among which, enhancements at the middleware layer and the network layer provide control functions and performance parameters to transport layer functions. Once a loss event is detected, the congestion control algorithm manager at the middleware layer will be informed and requested for commands. Using the global knowledge of the network, such as whether or not the receiver is ECN compatible, the congestion control algorithm manager will issue commands about which loss treatment algorithm should be executed.

We have proposed speculation algorithms at the transport

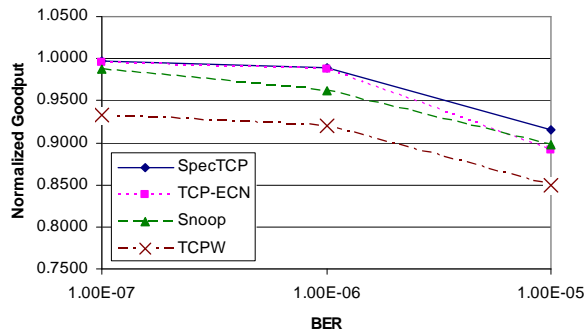


Fig. 15. Comparison of normalized Goodput.

layer, consisting of a conditional Bernoulli predictor used to predict the type of a loss event, and a speculative congestion control algorithm *SpecTCP* used to adjust the window size for the predicted losses.

To maximize the speculation accuracy, we have developed and evaluated mathematical models used to minimize congestion losses at ECN-capable RED gateways. This has been critical to the function of conditional Bernoulli predictor at the transport layer. Simulation-based verification results have shown that our mathematical models can effectively adjust RED gateway parameters so that congestion losses can be minimized. Most significantly, the buffer size inside a RED gateway based on our models are much smaller than previously suggested values. This means that the task itself has significant contribution to the network performance improvement.

We have analyzed the feasibility of this system. We have found that it is effective in improving network throughput over lossy links, capable of handling incorrect speculations, fair when used with other competing flows, backward compatible with legacy networks, and relatively easy to implement. The proposed system has been found to significantly improve throughput over lossy links. We have achieved significant throughput improvement, up to five times, over the TCP-Reno with ECN for data transfer across a typical satellite link with high BERs. Results have also shown that, the proposed system outperforms Snoop and TCP-Westwood which do not require ECN-capable RED gateways and end systems, and cross-layer architecture.

The work can be extended in the following ways. First of all, congestion loss minimization models can be extended by integrating adaptive control scheme to allow dynamic model computations and real-time gateway parameter reconfigurations, for the changes of network performance and environments. This may further improve the performance as the congestion loss minimization models can dynamically capture network variations, such as, nodes joining and leaving the network. Secondly, to validate the proposed system in a real environment, the proposed system may be implemented and tested with real Internet traffic. Thirdly, the function of congestion control manager at the middleware layer may be extended to take information from LINK and PHY layers, so that any improvements of the wireless link reliability at LINK

and PHY layers can be considered in making predictions by the conditional Bernoulli predictor.

## REFERENCES

- [1] H. Bai, S. Fu, and M. Atiquzzaman, "Transport layer design in mobile wireless networks," in *Inited Book Chapter in Design and Analysis of Wireless Networks*, Y. Pan and Y. Xiao, Eds. Nova Science Publishers, 2005.
- [2] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed. Morgan Kaufmann, 2003.
- [3] H. Bai and M. Atiquzzaman, "Error modeling schemes for fading channels in wireless communications: A survey," *IEEE Communications Surveys and Tutorials*, vol. 5, no. 2, pp. 2–9, October 2003.
- [4] S. Dawkins, G. Montenegro, M. Kojo, V. Magret, and N. Vaidya, "End-to-end performance implications of links with errors," RFC 3155, August 2001.
- [5] S. Cen, P. C. Cosman, and G. M. Voelker, "End-to-end differentiation of congestion and wireless losses," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 703–717, October 2003.
- [6] Y. Chen, R. Sendag, and D. Lilja, "Using incorrect speculation to prefetch data in a concurrent multithreaded processor," in *17th International Parallel and Distributed Processing Symposium*, Nice, France, April 2003.
- [7] S. Floyd, "TCP and explicit congestion notification," *ACM Computer Communication Review*, vol. 24, no. 5, pp. 10–23, October 1994.
- [8] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transaction on Networking*, vol. 1, pp. 397–413, August 1993.
- [9] B. Zheng and M. Atiquzzaman, "Active queue management in TCP/IP networks," in *High Performance TCP/IP Networking: Concepts, Issues, and Solutions*, M. Hassan and R. Jain, Eds. Prentice-Hall, 2004, pp. 281–307.
- [10] C. Liu and R. Jain, "Improving explicit congestion notification with the mark-front strategy," *Computer Networks*, vol. 35, no. 2-3, pp. 185–201, February 2001.
- [11] S. Kunniyur and R. Srikant, "End-to-end congestion control schemes: Utility functions, random losses and ECN marks," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 689–702, October 2003.
- [12] H. Bai and M. Atiquzzaman, "Using ECN marks to improve TCP performance over lossy links," in *Proc. First International Conference on E-Business and Telecommunication Networks*, Setubal, Portugal, August 2004, pp. 437–445.
- [13] S. Biaz and N. H. Vaidya, "Discriminating congestion losses from wireless losses using inter-arrival times at the receiver," in *IEEE ASSET Symposium*, Richardson, TX, USA, March 1999.
- [14] S. Biaz and N. Vaidya, "Discriminating congestion losses from wireless losses: A negative result," in *Seventh International Conference on Computer Communications and Networks*, New Orleans, LA, USA, October 1998.
- [15] K. Ramakrishnan and S. Floyd, "A proposal to add Explicit Congestion Notification (ECN) to IP," RFC 2481, January 1999.
- [16] T. Bonald, M. May, and J. Bolot, "Analytic evaluation of RED performance," in *INFOCOM*, Tel-Aviv, Israel, March 2000, pp. 1415–1424.
- [17] V. Misra, W. Gong, and D. Towsley, "Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED," in *ACM SIGCOMM*, Stockholm, Sweden, 2000, pp. 151–160.
- [18] A. Abouzeid and S. Roy, "Analytic understanding of RED gateways with multiple competing TCP flows," in *IEEE GLOBECOM*, San Francisco, CA, November 2000, pp. 555–560.
- [19] A. Abouzeid, S. Roy, and M. Azizoglu, "Stochastic modeling of tcp over lossy links," in *INFOCOM*, Tel Aviv, Israel, March 2000.
- [20] A. Mistra, T. Ott, and J. Baras, "The window distribution of multiple TCPs with random loss queues," in *IEEE GLOBECOM*, Rio de Janeiro, Brazil, December 1999, pp. 1714–1726.
- [21] H. Balakrishnan, S. Seshan, and R. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," *ACM Wireless Networks*, vol. 1, no. 4, pp. 469–481, December 1995.
- [22] C. Casetti, M. Gerla, S. Lee, M. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proceedings of ACM Mobicom*, Rome, Italy, July 2001, pp. 287–297.
- [23] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling*. John Wiley and Sons, 1991.