Exploiting the Predictability of TCP's Steady-state Behavior to Speed Up Network Simulation

Qi He, Mostafa Ammar, George Riley, Richard Fujimoto
College of Computing, Georgia Institute of Technology
Atlanta, GA 30332
qhe, ammar@cc.gatech.edu, riley@ece.gatech.edu, fujimoto@cc.gatech.edu

Abstract

In discrete-event network simulation, a significant portion of resources and computation are dedicated to the creation and processing of packet transmission events. For large-scale network simulations with a large number of high-speed data flows, the processing of packet events is the most time consuming aspect of the simulation. In this work we develop a technique that saves on the processing of packet events for TCP flows using the well established results showing that the average behavior of a TCP flow is predictable given a steady-state path condition. We exploit this to predict the average behavior of a TCP flow over a future period of time where steady-state conditions hold, thus allowing for a reduction (or elimination) of the processing required for packet events during this period. We consider two approaches to predicting TCP's steady-state behavior: using throughput formulas or by direct monitoring of a flow's throughput in a simulation. We design a simulation framework that provides the flexibility to incorporate this method of simulating TCP packet flows. Our goal is 1) to accommodate different network configurations, on/off flow behaviors and interaction between predicted flows and packet-based flows; and 2) to preserve the statistical behavior of every entity in the system, from hosts to routers to links, so as to maintain the accuracy of the network simulation as a whole. In order to illustrate the promise of this idea we implement it in the context of the ns2 *simulation system. A set of experiments illustrate the speedup and approximation of the simulation framework under different scenarios and for different network performance metrics.*

1. Introduction

In discrete-event network simulation, a significant portion of resources and computation are dedicated to the creation and processing of packet transmission events. For large-scale network simulations with a large number of high-speed data flows, the processing of packet events is the most time consuming aspect of the simulation. As a result any technique that would save on the processing of packet events can result in significant speedup of the simulation. In this work we develop a technique that saves on the processing of packet events for TCP flows. Needless to say, TCP traffic represents a large portion of the Internet traffic today and, therefore, large-scale network simulations typically involve the simulation of a large number of TCP packet flows.

It is well established[4, 5] that the average behavior of a TCP flow is predictable given a steady-state path condition. It is our goal in this paper to exploit this fact to predict the average behavior of a TCP flow over a future period of time where steady-state conditions hold, thus allowing for a reduction (or elimination) of the processing required for packet events during this period. In particular, given the current progress of a TCP data transfer (i.e., the highest sequence number sent or ACKed) and a predicted average throughput over a future steady-state period, we can predict the progress at a future time instance (e.g., we can predict the time a packet with a given sequence number will have been successfully delivered on average).

The TCP steady-state throughput formula[5] which estimates the average throughput based on round-trip time (RTT) and packet loss ratio has been shown to approximate the real TCP flow throughput closely. We observe that accurate measurement of RTT and loss ratio is quite manageable in a simulation system and the cost saving through throughput prediction could be significant as long as a flow is in steady-state for a long-enough time relative to the time needed to accurately measure loss ratio and RTT. Another way to predict TCP progress is simply to measure the average throughput in the steady-state for a while.

In this paper we design a simulation framework that provides the flexibility to incorporate this method of simulating TCP packet flows. The complexities of such a framework result from the requirements that: 1) it has to accommodate

Proceedings of the 10th IEEE Int'l Symp. on Modeling, Analysis, & Simulation of Computer & Telecommunications Systems (MASCOTS'02)
1526-7539/02 $17.00 © 2002 IEEE

different network configurations, on/off flow behaviors and interaction between predicted flows and packet-based flows; 2) it has to preserve the statistical behavior of every entity in the system, from hosts to routers to links, so as to maintain the accuracy of the network simulation as a whole.

Our work is related to two other simulation techniques: 1) use of fluid flow models and 2) use of *dead reckoning*. A number of researchers have reported good results by using the fluid flow models for network simulation [7, 8]These techniques tend to approximatenetwork flows as fluids and describe the flows and their interaction using flow equations. They are typically not derived from event-based simulation models (as our technique is) and as such become hard to integrate with existing event-based simulation approaches. Dead reckoning is another related technique by which the state of a simulated entity is estimated at some future time based on past history and some knowledge of the current state. This has been mostly used in the context of Distributed Interactive Simulation (DIS) (See for example [2, 3]). Our ideas explore related ideas in the specific context of network simulation and the simulation of TCP flows.

Our work has the following contributions:

- We designed a simulation framework that can utilize prediction to speed up simulations involving TCP flows, while maintaining the ability to simulate some flows with detailed packet events.
- We evaluate the speedup and approximation of the simulation framework. The two prediction techniques are compared for their approximation to packet-based simulation in different simulation configurations.

The rest of the paper is organized as follows. Section 2 details the design of the prediction-based simulation framework with predicted flows only. Section 3 describes two prediction techniques: formula-based prediction and direct measurement prediction. Section 4 extends the framework to accommodate the coexistence of packet-based and predicted flows. Section 5 evaluates the performance (speedup and approximation) with an *ns2*[1]implementation and compares the approximation of formula-based and direct measurement based predictions for different scenarios. This paper is concluded in Section 6.

2. Prediction-based Simulation Design

2.1. Simulation Context

To illustrate our simulation scheme, we use the following model to describe our simulation context. In this model, there are TCP flows and non-TCP flows. Each flow i ($1 \quad i \quad$) is an on-off flow which can be characterized

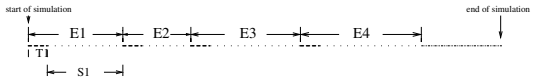

Figure 1. Simulation epochs

by a series of $(t_i \ , \ _i)$, $1 \quad = \quad k_i$, where t_i is the start of the th on-period of that flow and $_i$ is the number of bytes sent in that period. We call a TCP flow that is simulated with our proposed scheme a *predicted flow*, as compared to a normal flow which we call a *packet-based flow*. In this section, we will concentrate on predicted flows only, i.e., there is no non-TCP flow and all the TCP flows are predicted flows. In section 4, we will investigate cases where predicted flows interact with packet-based flows.

2.2. Simulation Framework Details

During a simulation, the system undergoes a series of epochs $_i$ as illustrated by Figure 1, where each epoch consists of a *transient period* (T, bold dashed segments) and a *steady-state period* (S, regular dotted segments). In this subsection, we describe the simulation system details of an epoch, including the switch from transient period to steady-state period, and the transition between successive epochs.

Epochs An epoch represents a certain snapshot configuration of active flows in the system. Transition from one epoch to another is triggered by changes of active flows, which we will detail shortly.

During the initial period of an epoch, TCP flows experience changes in average RTT and loss ratio due to the changes of competing flows, thus we call this a *transient period*. It is during this period that each TCP flow learns its average throughput. Hence all flows in our simulation system send packets during the period. After the transient period, the system switches to the steady-state period, during which each flow is assumed stable and saves processing by not scheduling any packet events. The length of the transient period should be chosen such that the average throughput seen by each flow has become relatively stable by the end of the transient period. It should be at least longer than , which is defined in [6] as the length of the initial slow start period of a new flow, as we will explain in section 3.

Intra-epoch Transition Intra-epoch transition is the transition from transient period to steady-state period within an epoch and involves two major actions.

One action is for the active TCP flows and the network to checkpoint their current states so that the next time the system switches to transient period, each of them has a starting state that statistically approximates the steady-state of this

epoch. TCP flow states include congestion window, RTT, retransmission timeout, and data transfer progress. The network state we checkpoint is the queue occupancy on each router. The checkpointing involves freezing the queue by not allowing any enqueue or dequeue operation, so that the same buffer occupancy can be restored at the next transition.

The other action is for active flows to predict the end of their current on-period. An active flow has a predicted average throughput r (bytes/sec) by the end of a transient period. Given that an active flow has bytes of data remaining to be sent during its current on-period, we can schedule a *steady-state period* event for the flow at /r , when the on-period is predicted to end. Since a TCP application can pass down additional data during an on-period, the amount of data to be sent within a flow's on-period is not fixed at the time of transition and we need to reschedule the steady-state period event whenever more data is buffered by TCP.

Inter-epoch Transition Transitions between epochs are triggered by changes of active flows. There are two kinds of event that result in such changes.

- end of an on-period. The end of a flow's on-period is usually indicated by the execution of its steady-state period event[1]. End of a flow is a special case of this.
- start of an on-period. The start of a flow's on-period is detected when new application data arrives during its off-period. Start of a new flow is a special case of this.

Assume that there are active flows and the steady-state period event of the ith active flow is scheduled at $_i$. Further assume that there are inactive flows and the th inactive flow will become active at . The next inter-epoch transition will then take place at:

$$i \ (\ _1 \ \ _2 \ \ _3 \ \ldots \ \ \ _1 \ \ _2 \ \ _3 \ \ldots \ ) \qquad (1)$$

However, neither $_i$ (1 $= i =$) nor (1 $=$ $=$) can be predicted, so that the next epoch transition can only be triggered dynamically as $_i$'s expire or as 's take place, rather than scheduled. For inter-epoch transitions, transitions from steady state to transient state, we need to cancel all the scheduled steady-state period events. Also, the active flows and the network need to recover/update states and resume sending/forwarding packets.

When a flow switches from steady state to transient state, it restores all the previously stored TCP states except those related to the progress of data transfer, which need additional attention. In particular, assume that at the time of the last intra-epoch transition, the highest sequence number reliably received by the other end ($i \ e \ t\_ack$) is a , the current $i \ e \ t\_ack$ is updated as follows: $i \ e \ t\_ack =$ $a \ r \ d \ r$, where r and $d \ r$ are the predicted throughput and the duration of the steady state in the last epoch. It is also important to restore the steady-state ACK locking behavior of the TCP flow, which we achieve by setting some state variables of the other end of the TCP connection. To restore the state of the path, we simply unfreeze each queue. When the queues resume operations, residual packets frozen in the queues during the last transition are delivered although will be discarded as obsolete because the receiving TCPs' states have been updated.

[1]There are cases where an on-period ends during the transient period.

3. Predicting Steady-state Throughput

We have described the framework to speed up TCP flow simulation with TCP throughput prediction. This section is dedicated to describing two techniques to predict the average throughput of a steady-state TCP flow.

3.1. Summary of TCP Behavior

A TCP flow can be characterized as an *initial slow start* phase followed by a *steady-state* phase [5, 6]. TCP tries to capture the maximum available bandwidth during the initial slow start phase with exponential increase of congestion window on each ACK. After the end of slow start, which is detected by the first packet loss, and the following timeout period, TCP goes into the steady-state. Steady-state of a TCP flow comprises of a series of timeout periods (*TO*), each of which is, in turn, composed of a *slow start* phase and multiple triple duplicate periods (*TDP*). The end of a TDP is marked by triple duplicate ACKs and the end of a TO is marked by a retransmission timeout event. Both the slow start phase and the TDP periods strictly follow certain algorithms to adapt the congestion window. Given the deterministic algorithm used by TCP in its steady-state, TCP behavior is very predictable. In particular, the length of the slow start phase and the length/number of TDPs within a TO, as well as the amount of data transfered during a TO can all be statistically determined.

3.2. Measurement Period

Larger speedup can be obtained by using shorter transient periods. However, a transient period should be long enough to capture the steady-state behavior of each flow.

Prediction should also be based on a period when no flow is still in its initial slow-start stage. We call a period during which the system is actually measuring (predicting) each flow's throughput a *measurement period*. It should begin after the initial slow-start phase of a newly (re)started flow in a transient period. The average length of an initial slow-start can be readily evaluated using recent results (see the expression for in [6]). Our experiments show that

a measurement period on the order of tens of TDPs gives close estimates of RTT and loss ratio.

3.3. Direct Throughput Measurement

One approach to predict is to directly measure the average throughput of each flow. To do that, we simply check the amount of data reliably delivered *during* the measurement period. We then derive the average throughput by dividing the amount of data by the length of the measurement period. The accuracy of this prediction depends on whether the measurement period is long enough for the predicted flow to get close to the limiting behavior of its steady-state.

3.4. Formula Based Prediction

Our proposed formula-based prediction is based on the model established in [5], where the authors characterize the steady state throughput of a TCP flow as a function of loss ratio, round trip time, b (number of segments ACKed by one ACK) and the maximum window size advertised by the receiver. The ($_{ax}$) formula applies to a bulk transfer TCP flow, i.e., a flow with an unlimited amount of data to send. Compared to previous work in [4], this model captures the timeout events in the steady state and hence the applicability of this equation is not limited to small packet loss ratio.

With formula-based prediction, we measure for each flow its average RTT and the packet loss probability on its path. We then estimate the average throughput with the throughput formula. The accuracy of the prediction depends both on the accuracy of the measured RTT and loss probability, and on the formula's approximation to real measurement. To obtain average RTT, we average all the RTT samples during a measurement period. Our experiments suggest that RTT measurement converges to the average RTT as the measurement period increases and a period of 10 TDPs is usually enough to ensure reasonable accuracy. During a measurement period, the simulation counts the number of packets received and dropped at every router a particular flow passes. The loss ratio of the flow is derived by adding these drop ratios[2]. Our experiments suggest that loss ratio converges at approximately the same rate as RTT.

4. Mixing Packet-based and Predicted Flows

Section 2.2 describes the simulation framework with all the flows being predicted TCP flows. The framework speeds up simulation through prediction and aggregation of packet events, i.e., the speedup comes at the cost of losing packet events. This is not desirable for simulations where every packet event is of interest to some flows. It is also possible that TCP flows are competing for bandwidth with non-TCP flows, for which the TCP steady-state prediction does not work. Both cases suggest the need to accommodate packet-based flows in the framework, which is the general case described in 2.1. To that end, we need only to make a few minor changes to the framework, as described below.

Changes to the Framework Within the new framework, the transition between epochs is triggered by the start/end of an on-period of any flow, including non-predicted packet-based flows. As of a predicted flow, the start of an on-period of a packet-based flow is also detected by observing the arrival of application data during an off-period. The end of an on-period in a packet-based flow, though, is always detected by the emptying of a sender buffer.

Accounting for Network Resource Usage The average RTT and loss ratio a flow experiences are affected by the queue limit and link bandwidth of on-path routers, as well as the additional queuing delay and losses due to competing flows. For each predicted flow, we can account for both the link bandwidth it uses and the queuing delay it causes by adjusting the bandwidth of each link on its path according to its share of the link's throughput. In particular, during a measurement period, we measure the throughput of each link on the flow's path in addition to predicting the flow's average throughput. Suppose that the estimate of the flow's throughput is r , the bandwidth and the measured throughput of a link are and r , respectively. When the flow switches to steady-state, we adjust the link bandwidth to $(1 - r\ /r\ )$. The proportionally reduced throughput accounts for the queuing delay (seen by competing flows) due to that flow, which in turn, accounts for the additional losses other flows will experience, given the same queue size. When a predicted flow switches back to transient state, the link bandwidth is increased according to the flow's share of throughput in the previous epoch.

5. Experimental Results

We implement the proposed framework and prediction schemes in *ns2*[1]. The predicted flow class inherits the TCP/FullTcp class and has the additional functionalities to switch, measure, predict and identify the changes between on and off periods. We also derive a new class of scheduler that handles all network changes.We conduct experiments to primarily answer two categories of question:

1. how much the predicted flow speeds up the simulation;
2. how closely the predicted flow approximates the packet-based flow, in terms of:

[2]Assume that the loss ratio of the on-path routers are p_1, p_2, ..., p_k, the loss ratio of the flow is $\sum_{i=1}^{k} p_i * (1 - \sum_{j=1}^{i-1} p_j)$.

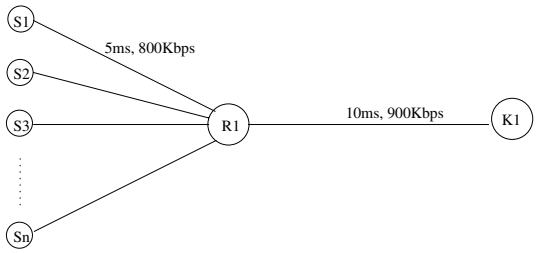

Figure 2. Topology I

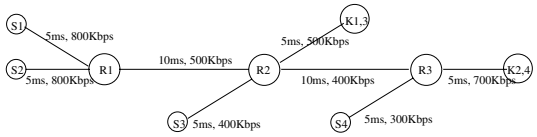

Figure 3. Topology II

(a) average flow throughput (inversely proportional to completion time)

(b) average network buffer occupancy;

(c) packet sequence traces of packet-based flows.

Throughout the experiments, flows perform bulk transfers that are never application-limited, but the system's behavior with on-off flows is easily derived from our experiments with multiple flows that start/finish at different times. Both prediction techniques are tested and compared whenever they might potentially make differences.

Most experiments use the network shown in Figure 2. The <propagation delay, bandwidth> on each link $_i$– $_1$ is <5ms, 800Kbps> and the queue size on $_1$ is 50, unless otherwise noted. There is a TCP connection and an FTP application between each $_i$ and $_1$. The network in Figure 3 is used in some experiments. The link parameters are shown in the figure.

Note that our simulation experiments use relatively low bit-rates but we use standard packet sizes of 536 bytes. We will report our results in the rest of this section using "seconds" as our time units. Ultimately what is most important is the relative packet transmission time and not the absolute value of time. While using seconds simplifies the discussion, it should be noted that more realistic bit rates (in the 10's of Mbps) would make our conclusions apply to much smaller time units in the 10's or 100's of milliseconds.

5.1. Speedup

From the description of the two predictions, we know that the prediction overhead is the same for both and they should achieve the same speedup. So we present only the results from the simulation of the formula-based prediction.

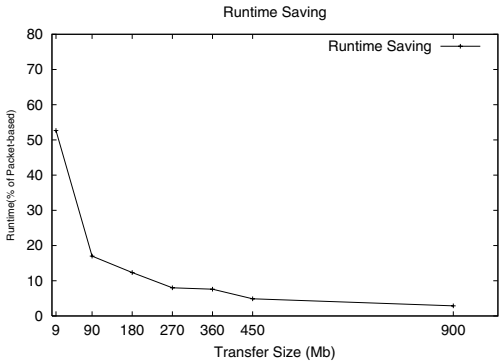

Figure 4. Speedup vs. transfer size

For the results in Figure 4, we use a FTP connection from $_1$ to $_1$ that transfers a file of different size in each test, and a 180-second measurement period. The average throughput is within a 3% difference from a packet-based simulation for each test result presented. Figure 4 plots the relative runtime of a prediction-based simulation to a packet-based simulation, against the transfer size of the TCP flow. The speedup with prediction based simulation increases dramatically with the transfer size, because the same cost of initial measurement is amortized over a longer steady-state period in a large transfer. Although this explanation assumes no additional network changes in a longer transfer, it is conceivable that a larger transfer benefits more as long as the number of network changes does not grow proportionally to the transfer size.

We also find that the relative runtime of the prediction-based simulation corresponds well to the proportion of time it spends measuring, suggesting that the cost of transitions and running in steady-state is negligible.

5.2. Approximation

In this subsection, we evaluate how the prediction-based simulation approximates the packet-based simulation, in terms of average flow throughput, network buffer occupancy and packet sequence traces. The choice of prediction scheme only affects the average throughput measurement, so comparison between the two predictions is only made in the evaluation of average throughput approximation.

5.2.1 Average Throughput

For a predicted TCP flow, we are primarily interested in evaluating its average throughput behavior (or completion time given a transfer size). We want to examine approxi-

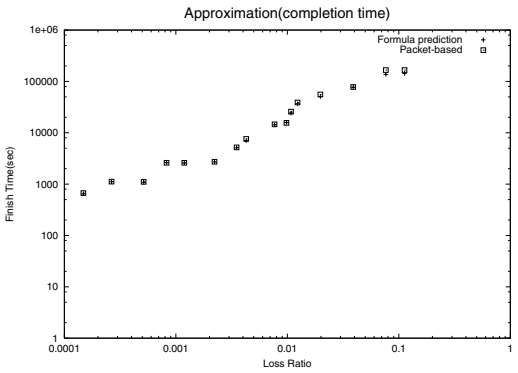

Figure 5. Approx vs. loss(formula)

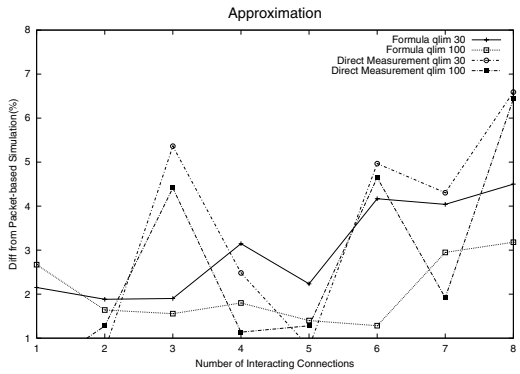

Figure 6. Approx vs. number of conns

mations for: 1) a range of RTT and loss ratio values; 2) situations where flows exhibit path heterogeneity in RTT and bottleneck bandwidth; 3) different number of flows.

Impact of Loss Ratio and RTT In this experiment, we use an FTP connection from $_1$ to $_1$ in Figure 2 and make the $_1$- $_1$ link a bottleneck. We choose different combinations of the queue limit on $_1$ and the bottleneck bandwidth such that we have a wide range of loss ratios and RTTs.

Figure 5 plots the approximation (difference from packet-based simulation) of completion time against loss ratio, of the formula-based prediction. It is seen that the approximation remains close for the range of loss ratios (up to 13%) we tested. Similar results (not shown here) is obtained for direct measurement prediction. However, there is a trend of increasing error with the formula-based prediction. By applying to the throughput formula the average loss ratio and RTT measured in the corresponding packet-based simulation, we are convinced that those larger errors in the formula-based simulation is caused by the error of the formula in those scenarios, which echoes the results in [5].

A similar set of experiments exploring the possible relation between RTT and approximation show that for different RTTs but with the same range of loss ratio, the approximation does not change significantly for either predictions.

Impact of Number of Interacting Flows When there are multiple connections sharing a path, the increased interaction between connections and the less predictable nature of loss ratio and RTT can stress the accuracy requirement of the simulation framework and the prediction techniques. For the results in Figure 6, we use the setting in Figure 2, where the FTP application on $_i$ starts sending in increasing order of i and the interval between two starts is 250 seconds. We take the approximation for each flow and use the average as the index of approximation of each test. Initially, we set the queue limit on $_1$ to 30. With the formula-based scheme, the approximation remains consistently good (below 4%) until there are 6 connections, and a better approximation is achieved after we change the queue limit to 100, which suggests that the increased error might be caused by increased loss ratio. Direct measurement does not show increasing error with more connections, nor is the approximation improved with increased queue size. These suggest again that there is no observable relationship between the loss ratio and the approximation of the direct measurement based prediction.

Bandwidth and Delay Heterogeneity The last two experiments in this category are designed to verify that the prediction scheme measures correctly for competing TCP flows that have different path characteristics, and that the simulation framework does not disturb the normal bandwidth sharing among them. We use differing bandwidths (delays) on links $_1-$ $_1$ and $_2-$ $_1$ and evaluate the approximation with varying differences.

The first set of experiments show that there is no global trend in the change of error as the upstream bandwidth difference between competing flows changes. However, as we experiment with competing flows with different RTTs, we find noticeable discrepancies for some situations, while still obtain a reasonable approximation in some other situations. These situations differ in their transfer size. We will explain these phenomena in this subsection.

We use the network shown in Figure 3 with two FTP connections, from $_1$ and $_2$, respectively. In the first configuration, $_1$ sends to $_1$ and $_2$ sends to $_2$; in the second configuration, both $_1$ and $_2$ send to $_1$. The two FTPs have different RTTs in the first case and same RTTs in the second. We first run packet-based simulation and measure the average throughput for each flow periodically. We find that it takes significantly longer for the average throughput to converge in the first configuration than in the second configuration, suggesting that for configurations where compet-

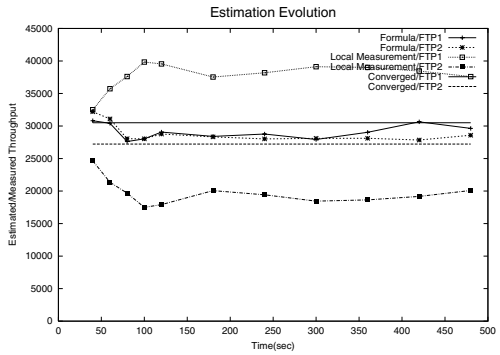

Figure 7. Throughput approximation(1)

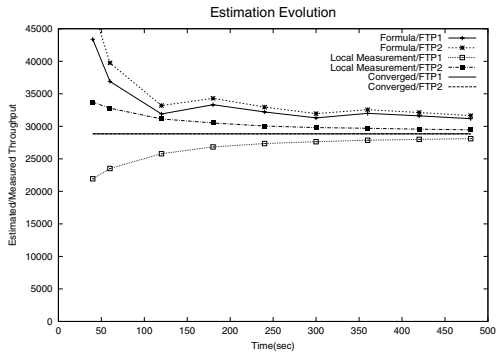

Figure 8. Throughput approximation(2)

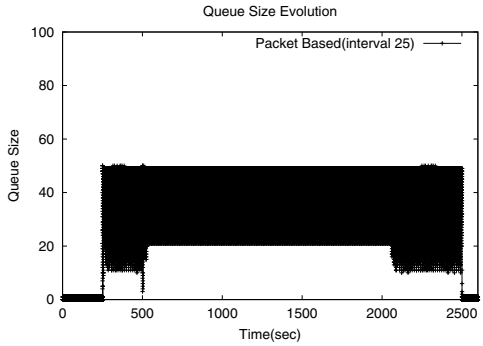

Figure 9. Queue occupancy–Packet-based

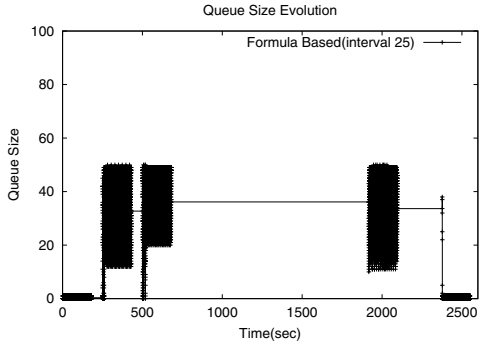

Figure 10. Queue occupancy–Formula-based

ing flows display significant heterogeneity in RTTs, direct measurement based prediction will only work for sessions that last a relatively short period.

Comparing the approximations by the two predictions for the two settings, in Figures 7 and 8 respectively, we observe that: 1) For *slowly converging* configurations like Figure 7, direct measurement prediction works only for very short flows; formula-based prediction works for longer-lived flows, but might not work well for short flows. There are chances that neither of them works well. 2) For *fast converging* settings like Figure 8, both predictions approximate well with a short measurement period. However, given such a measurement period,direct measurement renders a smoother and more accurate approximation curve.

5.2.2 Buffer Occupancy

From the network point of view, buffer occupancy is an interesting performance metric. We want to examine whether and how we can obtain a good approximation of this metric. One straightforward way is to measure the average occupancy of a queue when the path it belongs to is in a measurement period and take the measured average as the average buffer occupancy during the steady-state until the next transient period. The following experiment tests whether the period-wise averages match the measured average buffer occupancy: we use 3 FTP flows in Figure 2 and start each at intervals of 250 seconds. We set the measurement period to 180 seconds and monitor the buffer on 1.

Figures 9 and 10 show the instantaneous queue length sampled at an interval of every 25 changes. In the packet-based simulation, we can see several distinct periods where the buffer occupancy range is stable and those are the epochs in our simulation context. Transitions between two consecutive epochs correspond to changes of competing flows. In the prediction-based simulation, a period with constant queue length and a period with fluctuating queue length corresponds to the steady-state period and the transient period of an epoch, respectively. It can be seen that each epoch in Figure 9 matches a transient period followed by a steady-state period in Figure 10. The average buffer occupancy over the whole simulation are 30.88 and 29.49, respectively, leading to a difference of 4.5%.

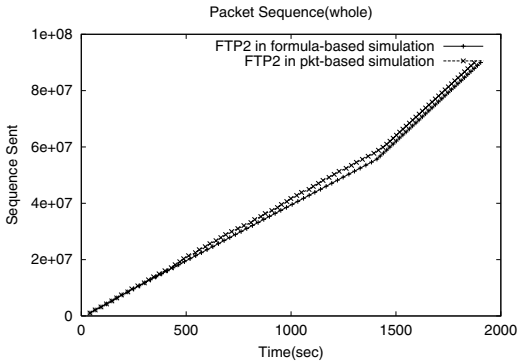

Figure 11. Packet sequence(whole trace)

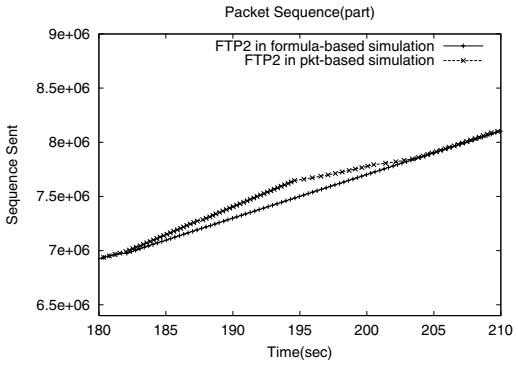

Figure 12. Packet sequence(after a transition)

5.2.3 Packet Sequence Traces

Packet sequence may be interesting to a packet-based flow. Due to the statistical nature of our approach in determining the instantaneous states during the inter-epoch transition,we cannot preserve the exact packet sequence. Our experiment is to validate that the average throughput of a packet-based flow which shares path with a predicted flow is close to what it experiences when the other flow is simulated with packet-based technique. We use two connections in Figure 2 but set link $_{2}$- $_{1}$ to 600Kbps. In the prediction based experiment, FTP2 uses a packet-based TCP connection and FTP1 uses a predicted TCP connection. Both flows start at time t=0. Figure 11 shows that the sequence number traces of FTP2 in a prediction based simulation and in a packet-based simulation match closely. Figure 12 is part of the FTP2 trace right after FTP1 switches to steady-state. We find that FTP2 initially gets to send faster, since the packets in the first congestion window (after FTP1's switch) experience lower delay and loss ratio than before, which increases the sender's congestion window faster. However, due to the reduced bandwidth after we account for FTP1's resource usage, RTT and loss ratio, hence throughput, soon return to the state before FTP1's switch.

6. Concluding Remarks

This paper proposes to speed up the simulation of TCP flows by predicting and aggregating packet events based on the predictability of steady-state TCP behavior. We design a prediction based simulation framework to use predicted flows and describe two techniques to predict the average throughput of a steady-state TCP flow, the direct measurement based prediction and the TCP throughput formula based prediction. Our results show that the prediction based simulation achieves significant speedup over packet-based simulation and the speedup increases dramatically with the transfer size. Both prediction techniques approximate the packet-based simulation reasonably well under all simulation setups except when competing flows display significant difference in RTT. Our analysis shows that the discrepancy results from the much slower convergence of average throughput in this case. Our future work will explore some estimation techniques (e.g., bandwidth estimate smoothing with faster convergence) that will be able to compensate for this effect. Our evaluations also demonstrate that in the prediction based simulation, the network and packet-based flows all experience behaviors that are close to what they will experience in packet-based simulations.

References

[1] Network Simulator http://www.isi.edu/nsnam/ns.

[2] M. A. Bassiouni, M. Chiu, M. Loper, M. Garnsey, and J. Williams. Performance and reliability analysis of relevance filtering for scalable distributed interactive simulation. *TOMACS*, 7, 1997.

[3] W. Cai, F. B. S. Lee, and L. Chen. An auto-adaptive dead reckoning algorithm for distributed interactive simulation. In *13th Workshop on Parallel and Distributed Simulation*, 1999.

[4] J.Mahdavi and S. Floyd. TCP-Friendly Unicast Rate-Based Flow Control. Sent to end2end-interest mailing list, Jan 1997.

[5] J.Padhye, V.Firoiu, D.Towsley, and J.Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *SIGCOMM*, 1998.

[6] N.Cardwell, S.Savage, and T.Anderson. Modeling TCP Latency. In *INFOCOMM*, 2000.

[7] D. Ros and R. Marie. Estimation of end-to-end delay in high-speed networks by means of fluid model simulations. In *13th European Simulation Multiconference*, 1999.

[8] A. Yan and W. Gong. Time–driven fluid simulation for high-speed networks. *IEEE Transactions on Information Theory*, 45(5):1588–1599, July 1999.