Exact Sampling of TCP Window States*

Ashish Goel †
University of Southern California

Michael Mitzenmacher ‡
Harvard University

February 15, 2002

Abstract

We demonstrate how to apply Coupling from the Past, a simulation technique for exact sampling, to Markov chains based on TCP variants. This approach provides a new, statistically sound paradigm for network simulations: instead of simulating a protocol over long times, or explicitly finding the stationary distribution of a Markov chain, use Coupling from the Past to quickly obtain samples from the stationary distribution.

Coupling from the Past is most efficient when the underlying state space satisfies a partial order and certain monotonicity conditions. To efficiently apply this general paradigm to TCP, we demonstrate that the states of a simple TCP model possess a monotonic partial order; this order appears interesting in its own right.

Preliminary simulation results indicate that this approach is quite efficient, and produces results which are similar to those obtained by simulating a TCP-Tahoe connection.

1 Introduction

There are two commonly used methods for determining or comparing the performance of TCP variants. The first approach is to use simulations over large time scales, using tools such as *ns* [17]. While useful in practice, this approach generally lacks a statistical basis, without a priori knowledge of how long a TCP simulation should run in order to obtain a good sample. A second approach is to develop a simplified TCP model, often based on a Markov chain. (Another recently proposed related approach is to use stochastic fluid models [15].) If enough simplifying assumptions are made, such a model may yield an equation (or bounding equations) for relevant quantities such as throughput [16, 19, 6, 3]. Such a tack often requires fairly extreme simplifications, however. Alternatively, given an appropriate model one may be able to calculate explicitly the equilibrium distribution [18, 22], from which relevant quantities can be derived. The calculations required, however, grow with the complexity of the model. For example, if complex loss models are used, calculating the equilibrium distribution may require significant resources.

We suggest another approach that may prove useful for studying performance of TCP variations or simplifications. This approach is to treat a TCP connection as a Markov chain and obtain a sample from the

*A preliminary version of this paper will appear in the Proceedings of IEEE Infocom, 2002.

†Ashish Goel is at the Department of Computer Science, University of Southern California. Supported in part by the DARPA NMS program under contract no. N66001-00-C-8066 ("SAMAN"). Email: agoel@cs.usc.edu

‡Michael Mitzenmacher is at the Division of Engineering and Applied Sciences, Harvard University. Supported in part by an Alfred P. Sloan Research Fellowship and NSF grants CCR-9983832, CCR-0118701, and CCR-0121154. Email: michaelm@eecs.harvard.edu

stationary distribution of this chain. Unlike previous work, our approach does not require computing the entire stationary distribution. Also, unlike simulation attacks that simply run the chain for a long period of time, our approach is grounded with a solid statistical basis. Specifically, under certain conditions, one can run a Markov chain in such a way that one is sure to obtain an *exact* sample from the stationary distribution.

We apply "Coupling from the Past" (CFTP) [20], a simulation technique widely used to sample combinatorial structures in mathematics and physics, to Markov chains for TCP. While this connection is theoretically interesting in its own right, we believe that CFTP may also prove a practically useful tool for network analysis. Indeed, we suspect that CFTP may prove useful for studying other similar complex network phenomena that can be modeled effectively as Markov chains.

CFTP is essentially a variant of the Markov Chain Monte Carlo method [8, 13] and a natural extension of the work on approximate and exact sampling for specific Markov chains [4, 14, 13]. Markov Chain Monte Carlo methods have already been used widely for problems arising from combinatorics (e.g. [5, 1]), physics (e.g. [10, 11]), statistics (e.g. [4]), and optimization (e.g. [7]). The reader is referred to an excellent description of CFTP by Propp and Wilson [20] for more detail.

Coupling from the Past is most effective when there is a partial order on the underlying state space with a monotone structure i.e. if $X \leq Y$ in the partial order, then this relationship is preserved as the states X and Y evolve in time. Accordingly, we first present a partial order defined on all possible states of a TCP connection. We show that this partial order results in a minimum and a maximum state. The minimum state corresponds to a connection performing slow start, with the slow start threshold and the congestion window set to the smallest possible values. The maximum state corresponds to a connection performing congestion avoidance, with the congestion window set to the maximum possible value. We then demonstrate that if we start in two different states which are ordered and couple their packet loss events, then the ordering is preserved as the states evolve. Thus the partial order has a monotonic structure. This allows us to apply CFTP to efficiently obtain an *exact* sample from the stationary distribution of the window sizes of a bulk TCP connection; here the stationary distribution is as seen by a random packet.

The sampling algorithm is very simple. For notational convenience, it is simplest to think of arranging matters so that packet sequence numbers are increasing but non-positive, so that our exact sample is obtained at the packet numbered 0. To obtain a sample from the exact distribution of the TCP Markov chain, we first generate an infinite packet loss pattern going backwards from packet 0,[1] and choose a small value τ. Then we simulate the TCP connection for packets numbered $-\tau$ to 0 starting in both the minimum and the maximum states. If the two states have converged by the time we get to the packet number 0, then the common state at the end is the desired sample. Otherwise, we double τ and repeat. Note that the work involved is just to run the Markov chains, albeit from two states instead of just one.

In order for the CFTP paradigm to apply, the underlying Markov chain needs to be ergodic. In the case of TCP, whether the chain is ergodic or not depends on the nature of the loss process that determines whether each packet gets dropped or successfully transmitted. Two interesting loss processes that result in ergodicity of the TCP chain are where the packet drops are i.i.d., and where the packet drops form a Markovian On-Off process. A further condition is that we need to be able to sample from the stationary distribution of the loss process, and create a loss pattern backwards in time. A more detailed explanation is presented in section 4.1.

We also perform a running time analysis of this scheme; in order to obtain one exact sample, we need to simulate a TCP connection over $O(N_{mix} \log W)$ packets on the average, where N_{mix} is the number of packets required for the TCP Markov chain to mix (see section 4.3 for a formal definition). It is unrealistic to expect to obtain any good sample (much less an exact sample) in fewer steps than the mixing time, and hence the running time guarantees are quite strong. Finally, we show how simple sub-sampling techniques

[1] Clearly an infinite loss pattern can not be generated in the traditional sense; in order to "generate" such a pattern, it suffices to fix a deterministic algorithm that takes a non-positive packet sequence number as an input, and output a bit indicating whether this packet got lost. We deal with this issue in greater detail in section 4.1.

can allow us to sample from the stationary distribution at a random time instant (as opposed to at a random packet departure epoch).

Several points about the CFTP approach are worth noting. First, even though the running time guarantee involves the quantity N_{mix}, the algorithm *does not need to know this quantity* to obtain the exact sample. This is a great asset, since computing N_{mix} or an upper bound on N_{mix} can be very complicated even for very simple Markov chains. Second, even though proving the correctness of the CFTP approach involves the partial order defined in section 2, the resulting sampling algorithm does not involve any knowledge of the partial order. Again, this is very useful since the partial order and the proof of monotonicity are quite intricate. Finally, it is important to note that CFTP does not strictly require monotonicity under a partial order, although these requirements aid analysis and greatly improve the practicality of using CFTP. Thus, although TCP variants can demonstrate non-monotonic behaviors [9], we believe this approach can be extended to Markov chains for other TCP variants besides the chain considered in this paper. Also, we believe that our monotonic partial order is interesting in its own right and may lead to further insights into the nature of TCP congestion control.

Our simulations of simple scenarios suggest that this approach is efficient, scales well with increasing maximum window sizes, and yields results which are close to those obtained by running the network simulator *ns* for TCP-Tahoe. Our simulation results are quite preliminary, and it would be interesting to develop a more extensive simulation infrastructure to explore the practical utility of the ideas in this paper.

In this paper, we restrict ourselves to bulk TCP connections, so we ignore the connection establishment phase. Further we assume that the TCP slow start and congestion avoidance algorithms are in place, but fast retransmit and fast recovery algorithms are not (see [12, 21, 2] for a detailed description of these algorithms). Extending our approach to all variants of TCP and to other networking protocols is an interesting open problem.

Section 2 defines the partial order, and section 3 proves that this partial order is monotonic. Section 4 details how the CFTP paradigm can be applied to this problem, section 5 presents the simulation results, and section 6 concludes the paper.

2 A Partial Order on TCP Windows

In this section, we define a simplified state space for TCP, and provide a partial order on this state space. We show that this partial order has unique minimum and maximum elements, which is useful in applying CFTP.

Definition 1 *Given a TCP connection $\mathcal{C}$, the state $S(\mathcal{C})$ of the TCP window is the triplet* $\langle$MODE, SSTHRESH, CWND$\rangle$ *where*

- MODE $=$ SS *if the TCP connection is performing slow start, and* MODE $=$ CA *if the connection is performing congestion avoidance.*
- SSTHRESH *denotes the slow start threshold. If* MODE $=$ CA *then the slow start threshold is irrelevant and is denoted by the symbol $\emptyset$.*
- CWND *denotes the congestion window of the TCP connection.*

We use MODE(S), SSTHRESH(S), and CWND(S) to denote the three components of the state S; also we overload notation and use MODE$(\mathcal{C})$ etc. to denote MODE$(S(\mathcal{C}))$ etc.

Definition 2 *Let $\mathcal{S}$ denote the space of all valid TCP states.*

We do not define formally what the valid TCP states are, but invoke the properties of TCP as and when needed to disqualify states from belonging to $\mathcal{S}$. In particular, we assume that a TCP connection goes out of slow start and into congestion avoidance as soon as the congestion window becomes equal to or exceeds the slow start threshold.

We now define a relation $\prec$ on $\mathcal{S}$.

Definition 3 *Given two states $A, B \in \mathcal{S}$, the relation $A \prec B$ holds if and only if exactly one of the following is true:*

1. $\text{MODE}(A) = \text{MODE}(B) = \mathsf{CA}$ *and* $\text{CWND}(A) \leq \text{CWND}(B)$
2. $\text{MODE}(A) = \text{MODE}(B) = \mathsf{SS}$, $\text{SSTHRESH}(A) \leq \text{SSTHRESH}(B)$, *and* $\text{CWND}(A) \leq \text{CWND}(B)$
3. $\text{MODE}(A) = \mathsf{SS}$, $\text{MODE}(B) = \mathsf{CA}$, *and* $\text{SSTHRESH}(A) \leq \text{CWND}(B)$
4. $\text{MODE}(A) = \mathsf{CA}$, $\text{MODE}(B) = \mathsf{SS}$, *and* $\text{CWND}(A) \leq \text{CWND}(B)$

This relation has some very interesting properties. In particular, we will show that this relation is a partial order. Figure 1 illustrates this partial order for a simple case.

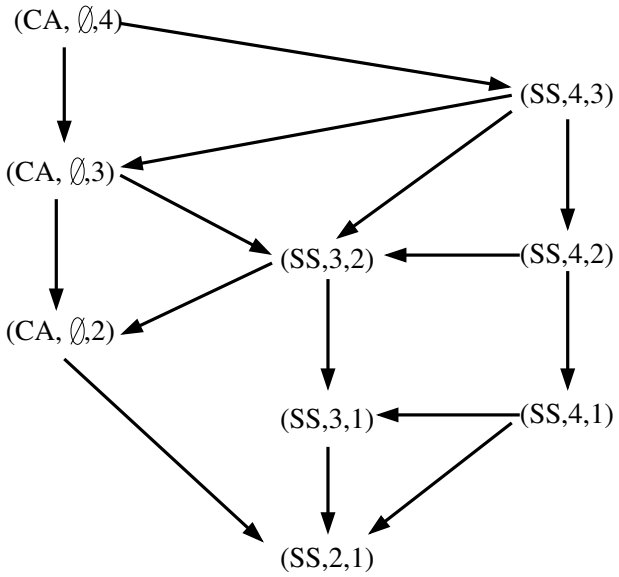

Figure 1: A pictorial representation of the partial order $\prec$ for a small set of TCP window states. There is a directed path from B to A in the above graph iff $A \prec B$. Notice that there is a minimum state $(\mathsf{SS}, 2, 1)$, a maximum state $(\mathsf{CA}, \emptyset, 4)$, and that there are incomparable states (eg. the states $(\mathsf{CA}, \emptyset, 3)$ and $(\mathsf{SS}, 4, 2)$).

2.1 The partial order property

We now claim that the relation $\prec$ is a partial order i.e. it is reflexive, anti-symmetric, and transitive.

Theorem 1 *The relation $\prec$ is a partial order.*

Proof: $X \prec X$ is trivial to prove, so reflexivity holds.

To prove anti-symmetry, we need to show that if $X \neq Y$ then $X \prec Y \Rightarrow Y \not\prec X$. The proof is by contradiction. Suppose $X \neq Y$, $X \prec Y$ and $Y \prec X$. We consider four cases:

1. Suppose $\text{MODE}(X) = \text{MODE}(Y) = \mathsf{CA}$. Then, combining $X \prec Y$ with rule 1 in definition 3, we know that $\text{CWND}(X) \leq \text{CWND}(Y)$. Since $Y \prec X$ also holds, we know that $\text{CWND}(Y) \leq \text{CWND}(X)$ i.e. $\text{CWND}(X) = \text{CWND}(Y)$. Since $\text{MODE}(X) = \text{MODE}(Y) = \mathsf{CA}$, we know that $\text{SSTHRESH}(X) = \text{SSTHRESH}(Y) = \emptyset$. Hence $X = Y$ which is a contradiction.

2. Suppose $\text{MODE}(X) = \text{MODE}(Y) = \mathsf{SS}$. Then combining rule 2 in definition 3 with $X \prec Y$ and $Y \prec X$, we get $\text{SSTHRESH}(X) \leq \text{SSTHRESH}(Y)$, $\text{SSTHRESH}(Y) \leq \text{SSTHRESH}(X)$,$\text{CWND}(X) \leq \text{CWND}(Y)$, and $\text{CWND}(Y) \leq \text{CWND}(X)$ simultaneously. Together, these imply that $X = Y$, which is a contradiction.

3. Suppose $\text{MODE}(X) = \mathsf{SS}$ and $\text{MODE}(Y) = \mathsf{CA}$. Then by rule 3 and the fact that $X \prec Y$, we get $\text{SSTHRESH}(X) \leq \text{CWND}(Y)$. Combining rule 4 with $Y \prec X$, we get $\text{CWND}(Y) \leq \text{CWND}(X)$. Together, the two imply that $\text{SSTHRESH}(X) \leq \text{CWND}(X)$. But TCP goes out of slow start and into congestion avoidance when the congestion window becomes equal to or exceeds the slow start threshold. Hence state X could not be in slow start mode, which is a contradiction.

4. Suppose $\text{MODE}(X) = \mathsf{CA}$ and $\text{MODE}(Y) = \mathsf{SS}$. This is symmetric with the previous case.

Since all four cases above result in a contradiction, we have established that $\prec$ is anti-symmetric.

We must now prove that $\prec$ is transitive i.e. $X \prec Y$ and $Y \prec Z$ together imply $X \prec Z$. Again, we divide the proof into four steps:

1. Suppose $\text{MODE}(X) = \text{MODE}(Y) = \mathsf{CA}$. If $\text{MODE}(Z) = \mathsf{CA}$, then by rule 1, we have $\text{CWND}(X) \leq \text{CWND}(Y) \leq \text{CWND}(Z)$ which implies that $\text{CWND}(X) \leq \text{CWND}(Z)$. Invoking rule 1 again, we have $X \prec Z$. If $\text{MODE}(Z) = \mathsf{SS}$, then by rules 1 and 4, we have $\text{CWND}(X) \leq \text{CWND}(Y) \leq \text{CWND}(Z)$; invoking rule 4 again implies that $X \prec Z$.

2. Suppose $\text{MODE}(X) = \text{MODE}(Y) = \mathsf{SS}$. If $\text{MODE}(Z) = \mathsf{SS}$ then by invoking rule 2, we know that $\text{CWND}(X) \leq \text{CWND}(Y) \leq \text{CWND}(Z)$ and $\text{SSTHRESH}(X) \leq \text{SSTHRESH}(Y) \leq \text{SSTHRESH}(Z)$; invoking rule 2 again, we obtain $X \prec Z$. If $\text{MODE}(Z) = \mathsf{CA}$ then invoking rules 2 and 3, we get $\text{SSTHRESH}(X) \leq \text{SSTHRESH}(Y) \leq \text{CWND}(Z)$; reinvoking rule 3 gives $X \prec Z$.

3. Suppose $\text{MODE}(X) = \mathsf{SS}$ and $\text{MODE}(Y) = \mathsf{CA}$. If $\text{MODE}(Z) = \mathsf{CA}$ then invoking rules 3 and 1, we get $\text{SSTHRESH}(X) \leq \text{CWND}(Y) \leq \text{CWND}(Z)$; invoking rule 3 again gives $X \prec Z$. The case $\text{MODE}(Z) = \mathsf{SS}$ is a little more involved. Invoking rules 3 and 4, we get $\text{SSTHRESH}(X) \leq \text{CWND}(Y) \leq \text{CWND}(Z)$. This in itself is not enough to invoke rule 2 and claim that $X \prec Z$. But observe that since X and Z are both in slow start, we know that $\text{CWND}(X) < \text{SSTHRESH}(X)$ and $\text{CWND}(Z) < \text{SSTHRESH}(Z)$. Combining these two inequalities with $\text{SSTHRESH}(X) \leq \text{CWND}(Z)$ gives $\text{CWND}(X) < \text{SSTHRESH}(X) \leq \text{CWND}(Z) < \text{SSTHRESH}(Z)$. We can now invoke rule 2 to claim that $X \prec Z$.

4. Suppose $\text{MODE}(X) = \mathsf{CA}$ and $\text{MODE}(Y) = \mathsf{SS}$. If $\text{MODE}(Z) = \mathsf{SS}$, then rules 4 and 2 imply that $\text{CWND}(X) \leq \text{CWND}(Y) \leq \text{CWND}(Z)$; invoking rule 2 again gives $X \prec Z$. If $\text{MODE}(Z) = \mathsf{CA}$, then rule 4 implies that $\text{CWND}(X) \leq \text{CWND}(Y)$, $\text{MODE}(Y) = \mathsf{SS}$ implies that $\text{CWND}(Y) < \text{SSTHRESH}(Y)$, and rule 3 implies that $\text{SSTHRESH}(Y) \leq \text{CWND}(Z)$. Combining the above, we obtain $\text{CWND}(X) < \text{CWND}(Z)$; invoking rule 1 now gives us $X \prec Z$.

■

It is interesting to note that the partial order $\prec$ is a total order if restricted to only those states which are in the congestion avoidance mode. The above proof illustrates how the definition of $\prec$ is carefully tailored for us to be able to prove that the relation $\prec$ is a partial order. There are other partial orders that can be defined on the

TCP window state space. What makes the relation $\prec$ particularly interesting is the existence of a minimum and a maximum element.

2.2 The lower bound property

We now define a special "lower bound" state $\hat{L}$.

Definition 4 *The state $\hat{L}$ is $\langle \mathsf{SS}, 1, 1 \rangle$ i.e. the TCP connection is in slow start, and the congestion window and the slow start threshold[2] are both set to 1.*

The state $\hat{L}$ is a minimum i.e. $\hat{L} \prec X$ for all $X \in \mathcal{S}$. This is easily verified by looking at rules 2 and 3.

2.3 The upper bound property

We now define a special "upper bound" state $\hat{U}$. We use the terms max_cwnd and max_ssthresh to refer to the maximum possible window size and the maximum possible slow start threshold, respectively. These terms typically depend on the TCP variant in use, the advertised window size of the receiver, and the configuration of the end-hosts. We assume that max_ssthresh $\leq$ max_cwnd; if not then max_ssthresh can be set equal to max_cwnd without any change in TCP behavior. By a similar argument, we assume that max_cwndis no larger than the receiver's advertised congestion window.

Definition 5 *The state $\hat{U}$ is $\langle \mathsf{CA}, \emptyset, \mathit{max\_cwnd} \rangle$ i.e. the TCP connection is in the congestion avoidance phase and the congestion window is the maximum possible.*

The state $\hat{U}$ is a maximum i.e. $X \prec \hat{U}$ for all $X \in \mathcal{S}$. This is easy to verify by looking at rules 1 and 3.

3 Monotonicity in the TCP Window Space

We show here that the partial order $\prec$ has a nice monotonic property which allows us to apply Coupling from the Past, assuming the loss process satisfies certain useful properties.

Consider two valid states X and Y of a TCP connection $\mathcal{C}$ such that $X \prec Y$. Let Next(X) and Next(Y) denote the states of this connection after sending one packet each from states X and Y and receiving the corresponding ACK or NACK. Here we make the following simplifying assumption: when a loss occurs, it begins a *loss event* that causes all subsequent packets to be lost until a timeout occurs. Since we are not using fast retransmit or fast recovery, essentially this assumption provides a lower bound on TCP performance; we ignore packets that may have been received that and will be acknowledged later. Similar assumptions have been made in other work, e.g. [19, 3].

If we couple the fate of the next packet sent in state X with the fate of the next packet sent in state Y (i.e. either both begin a loss event, or both are successfully transmitted), then Next$(X) \prec$ Next(Y).

We recall how Next(X) depends on X.

1. If $\text{MODE}(X) = \mathsf{SS}$, then a successful transmission yields $\text{CWND}(\text{Next}(X)) = \text{CWND}(X) + 1$. Also $\text{MODE}(\text{Next}(X)) = \mathsf{CA}$ if $\text{CWND}(\text{Next}(X)) = \text{SSTHRESH}(X) = \text{SSTHRESH}(\text{Next}(X))$.
2. If $\text{MODE}(X) = \mathsf{CA}$, then a successful transmission yields $\text{CWND}(\text{Next}(X)) = \text{CWND}(X) + 1/\text{CWND}(X)$.
3. On a packet loss, $\text{MODE}(\text{Next}(X)) = \mathsf{SS}$, $\text{SSTHRESH}(\text{Next}(X)) = \max\{\lceil \text{CWND}(X)/2 \rceil, 2\}$, and $\text{CWND}(\text{Next}(X)) = 1$.

[2] Most variants of TCP set the congestion window to at least 2; for these variants we should define $\hat{L} = \langle \mathsf{SS}, 2, 1 \rangle$. All the results in this paper hold with this variation as well.

Before proving the theorem for the above setup, it is worth emphasizing that our approach could easily apply to other common TCP simplifications. For example, in some cases TCP is modeled without slow start; it is instead assumed that the process is always in congestion avoidance, and that a loss causes the sending window to shrink by some constant factor. (See, for example, [19, 3] for relevant discussions.) In this case our Markov chain state space would be even simpler (we could avoid the SS mode altogether), and we we could prove monotonicity in a manner similar to the theorem below.

Theorem 2 *If $X \prec Y$ then Next$(X) \prec$ Next(Y).*

Proof: As before, we require a careful case by case analysis:

1. Suppose $\text{MODE}(X) = \text{MODE}(Y) = \mathsf{CA}$. Then by rule 1 in definition 3, we know that $\text{CWND}(X) \leq \text{CWND}(Y)$. If there is no loss, then $\text{MODE}(\text{Next}(X)) = \text{MODE}(\text{Next}(Y)) = \mathsf{CA}$, $\text{CWND}(\text{Next}(X)) = \text{CWND}(X) + 1/\text{CWND}(X)$, and $\text{CWND}(\text{Next}(Y)) = \text{CWND}(Y) + 1/\text{CWND}(Y)$. Now $\text{CWND}(\text{Next}(X)) \leq \text{CWND}(\text{Next}(Y))$, as the function $f(x) = x + 1/x$ is increasing in x for $x \geq 1$. Hence $\text{CWND}(\text{Next}(X)) \leq \text{CWND}(\text{Next}(Y))$, so by rule 1 in definition 3, Next$(X) \prec$ Next(Y). If there is a loss, then $\text{MODE}(\text{Next}(X)) = \text{MODE}(\text{Next}(Y)) = \mathsf{SS}$, $\text{CWND}(\text{Next}(X)) = \text{CWND}(\text{Next}(Y)) = 1$, $\text{SSTHRESH}(\text{Next}(X)) = \max\{\lceil \text{CWND}(X)/2 \rceil, 2\}$, and $\text{SSTHRESH}(\text{Next}(Y)) = \max\{\lceil \text{CWND}(Y)/2 \rceil, 2\}$. Note $\text{SSTHRESH}(\text{Next}(X)) \leq \text{SSTHRESH}(\text{Next}(Y))$. By rule 2 in definition 3, Next$(X) \prec$ Next(Y).

2. Suppose $\text{MODE}(X) = \text{MODE}(Y) = \mathsf{SS}$. By rule 2 in definition 3 we have $\text{SSTHRESH}(X) \leq \text{SSTHRESH}(Y)$ and $\text{CWND}(X) \leq \text{CWND}(Y)$. If there is a loss, it is as in case 1. If there is no loss, then the SSTHRESH remain unchanged, $\text{CWND}(\text{Next}(X)) = \text{CWND}(X) + 1$, and $\text{CWND}(\text{Next}(Y)) = \text{CWND}(Y) + 1$, so $\text{CWND}(\text{Next}(X)) \leq \text{CWND}(\text{Next}(Y))$. If both states remain in mode SS , then by rule 2 in definition 3, Next$(X) \prec$ Next(Y). If both states move to mode CA , then by rule 1 in definition 3, Next$(X) \prec$ Next(Y). If state X moves to mode CA and Y does not, by rule 4 in definition 3, Next$(X) \prec$ Next(Y). If state Y moves to mode CA and X does not, then $\text{SSTHRESH}(X) \leq \text{SSTHRESH}(Y) \leq \text{CWND}(\text{Next}(X))$, so by rule 3 in definition 3, Next$(X) \prec$ Next(Y).

3. Suppose $\text{MODE}(X) = \mathsf{SS}$ and $\text{MODE}(Y) = \mathsf{CA}$. Then by rule 3, $\text{CWND}(X) < \text{SSTHRESH}(X) \leq \text{CWND}(Y)$. If there is a loss, it is as in case 1. If there is no loss, then $\text{SSTHRESH}(X)$ remains unchanged, $\text{CWND}(\text{Next}(X)) = \text{CWND}(X) + 1$, and $\text{CWND}(\text{Next}(Y)) = \text{CWND}(Y) + 1/\text{CWND}(Y)$. If $\text{MODE}(\text{Next}(X)) = \mathsf{SS}$, then we have $\text{CWND}(\text{Next}(X)) < \text{SSTHRESH}(X) < \text{CWND}(\text{Next}(Y))$, and by rule 3 in definition 3, Next$(X) \prec$ Next(Y). If $\text{MODE}(\text{Next}(X)) = \mathsf{CA}$, then $\text{CWND}(\text{Next}(X)) = \text{SSTHRESH}(X) < \text{CWND}(\text{Next}(Y))$, and by rule 1 in definition 3, Next$(X) \prec$ Next(Y).

4. Suppose $\text{MODE}(X) = \mathsf{CA}$ and $\text{MODE}(Y) = \mathsf{SS}$. Then by rule 4 in definition 3, $\text{CWND}(X) \leq \text{CWND}(Y)$. If there is a loss, it is as in case 1. If there is no loss, then $\text{CWND}(\text{Next}(X)) = \text{CWND}(X) + 1/\text{CWND}(X)$, and $\text{CWND}(\text{Next}(Y)) = \text{CWND}(Y) + 1$. If $\text{MODE}(\text{Next}(Y)) = \mathsf{SS}$, then by rule 4 in definition 3, Next$(X) \prec$ Next(Y); otherwise, it follows by rule 1.

■

4 Applying CFTP to the TCP Window Space

In this section, we demonstrate how to use our previous results to apply CFTP to the TCP window space. We confine ourselves to bulk TCP connections; specifically, we choose a position in the stream and label its packet sequence number as 0, and we assume there are an infinite sequence of packets prior to this one. Our goal is to find the state of the TCP connection as seen by this packet. All other packets have negative

numbers. Further, we require that each packet is labeled with a bit that indicates whether this packet was successfully transmitted by the network. A loss process is a stochastic process that produces this labeling of the infinite stream of packets.

In this paper, we are concerned with loss processes where packet drops are governed by a Markov chain. A loss process along with the TCP congestion control algorithms define a Markov chain that we call the TCP Markov chain. Note that in our TCP Markov chain, we do not include packets that are sent out after the dropped packet (until a timeout happens); hence our loss process really signals loss events.

4.1 Ergodic TCP loss processes

Definition 6 *A loss process is said to be an* ergodic TCP loss process *if the resulting TCP Markov chain is ergodic.*

One sufficient condition for a loss process to be an ergodic TCP loss process is that arbitrarily long sequences of packet drops and successful packet transmissions should occur with a non-zero probability. Two interesting loss processes that result in ergodicity of the TCP chain are where the packet drops are i.i.d, and where the packet drops form a Markovian On-Off process.

In order for the CFTP paradigm to apply, we need to be able to generate the loss pattern from $-\infty$ to 0. Since this is an infinite sequence, we can not enumerate it in any traditional sense; instead we need to generate elements of this sequence *on demand* and in a consistent fashion. For i.i.d. drops, it is easy to generate elements of this sequence on demand. For Markovian On-Off drops, we can sample from the stationary distribution of the On-Off process to determine whether packet 0 got dropped. We can then do a backwards walk in the On-Off Markov chain to generate other elements of this sequence on demand. Sampling from the stationary distribution of a Markovian On-Off process is easy. Similar techniques should apply for other natural loss processes.

4.2 Applying CFTP to sample from the TCP state space

We use the CFTP paradigm for monotonic Markov chains as defined by Propp and Wilson [20] to obtain a sample from the stationary distribution of TCP window sizes. The evolution of the states $\hat{L}$ and $\hat{U}$ in the Markov chain is simulated from event $-\tau$ to 0. If the two processes are in the same state at event 0, then this common state is output as the sample from the stationary distribution. If not, then τ is doubled and the process is repeated. Note that doubling τ is chosen for simplicity; any factor greater than one would do. (See [20] for details.) It is important to reuse the same random numbers at event $-t$ during all the iterations of the algorithm. In our scenario, this means that during all the iterations of the algorithm, we should have a consistent view of whether the packet numbered $-t$ was lost. The main intuition is that when the sample paths starting from the bottom state $\hat{L}$ and the top state $\hat{U}$ converge, they sandwich the entire state space in between. It is important to follow this procedure exactly as described and not cut corners such as simulating states $\hat{L}$ and $\hat{U}$ forward in time from event 0 till they converge. The reader is referred to an excellent description of the process by Propp and Wilson [20] for more detail.

The following theorem follows from a general theorem due to Propp and Wilson [20] regarding the correctness of the CFTP paradigm.

Theorem 3 *The algorithm outlined above samples* exactly *from the stationary distribution of the Markov chain defined by the TCP slow start and congestion avoidance mechanisms in conjunction with an ergodic TCP loss process.*

4.3 Running Time Analysis

We first need some definitions. Assume we are given an ergodic Markov chain M; further assume that the state space of this Markov chain is equipped with a partial order, a minimum state, a maximum state, and a monotonic property.

Definition 7 *Let $\bar{d}(k) = \max_{\pi_1,\pi_2} ||\pi_1^k - \pi_2^k||$ where π^k is the distribution governing the Markov chain M after k transitions, when started in a random state governed by the distribution π. The* mixing time $T_{mix}(M)$ *is defined to be the smallest k for which $\bar{d}(k) \leq 1/e$.*

The mixing time need not necessarily be a "time;" in fact, for the TCP Markov chain it is going to denote the number of packets transmitted.

Definition 8 *The* convergence time $T^*(M)$ *denotes the number of simulation steps required by the CFTP process to return a sample from the stationary distribution of M.*

Definition 9 *Given a partial order on the state of the Markov chain M, the* chain-length $C(M)$ *of M is the length of the largest ordered sequence of distinct states.*

The following general lemma was proved by Propp and Wilson.

Lemma 4 $\boldsymbol{E}[T^*(M)] \leq 2T_{mix}(M) \cdot (1 + \ln C(M))$.

Let W denote the maximum congestion window size of the TCP connection under study, and let N_{mix} denote the mixing time of the TCP Markov chain.

Lemma 5 *The number of distinct states in the TCP Markov chain is $O(W^3)$.*

Proof: We need to prove that $|\mathcal{S}| = O(W^3)$. If a TCP connection is in the slow start mode, then its slow start threshold and congestion window are both integers $\leq W$. Hence the number of different slow-start states is at most W^2. Now suppose that the TCP connection is in the congestion avoidance phase, and that the transition from slow start to congestion avoidance was made when the window size was x. There are at most W choices for the value of x. Also, each successful ack during congestion avoidance results in the window size being increased by at least $1/W$. Consequently, there can be at most W^2 distinct window sizes encountered as the window increases from x to W. Hence, there can be at most W^3 different states in the congestion avoidance phase. The total number of states is at most $W^2 + W^3 = O(W^3)$. ■

Since the chain-length can not be any larger than the number of states, we obtain the following theorem:

Theorem 6 *The convergence time for the TCP Markov chain is $O(N_{mix} \ln W)$.*

4.4 Sub-sampling to obtain samples at a random time

When the window size is large, a large number of packets see that window size. Thus the stationary distribution as seen by a random packet is biased towards larger window sizes compared to the stationary distribution at a random time instant. In order to obtain a sample from the latter distribution, we need to discard some of the samples returned by the CFTP process outlined above.

We use the term sampling interval to represent the larger of the timeout value and the RTT for the TCP connection (the timeout value is the duration after which the sender presumes that an unacknowledged packet has been dropped). We denote the sampling interval by I. To obtain samples at a random time we use the following sub-sampling algorithm:

1. Use CFTP to obtain a sample X from the stationary distribution of the TCP state space as seen by a random packet.

2. Simulate the TCP connection starting from X to determine the interval I' after which the next packet is sent.

3. With probability I'/I, output this sample and exit; else, go back to step 1.

The following theorem states that the sub-sampling algorithm is correct and efficient.

Theorem 7 *The sub-sampling algorithm outlined above returns an exact sample from the stationary distribution of the TCP Markov chain as seen at a random time instant. Further, the sub-sampling algorithm makes at most $W \cdot (I/RTT)$ calls to the CFTP algorithm on an average.*

Proof:

(a) Correctness: This part is relatively straightforward, so instead of giving a formal proof, we sketch the main intuition. Assume that the state of the TCP connection changes at packet-departure epochs. Now the state seen by a random packet persists till the next packet is sent out. This happens an interval I' later. Therefore we need to weight this sample by I'. Since I' must be less than I (recall that I is the larger of the RTT and the timeout), choosing to retain the sample with probability I'/I gives it the appropriate proportional weight.

(b) Running time: Let us artificially divide the entire packet sequence into chunks of W contiguous packets. Let us examine a specific chunk B. Let I'_1 denote the time between the first and second packet departures in the chunk, I'_2 denote the time between the second and third packets in the chunk, and so on. Further, let I'_W denote the time between the last packet departure from chunk B and the first packet departure from the next chunk. The CFTP algorithm in section 4.2 gives a sample from the stationary distribution as seen by a random packet. Let us condition our sample such that the random packet must belong to chunk B. Given this conditioning, each of the W packets on this chunk is chosen with probability $1/W$. Hence the probability that the sub-sample succeeds is $(1/W)\sum_{j=1}^{W} I'_j/I$. Since the chunk is of size W (i.e. the maximum window size), it requires at least RTT time to go across, which implies that $\sum_{j=1}^{W} I'_j \geq$ RTT. Therefore the success probability given this conditioning is at least $(1/W) \cdot (\text{RTT}/I)$. We did not use any special properties of the chunk B, so the same lower bound on the success probability holds if we remove the conditioning. It now follows that the expected number of samples required for success is at most $W \cdot (I/\text{RTT})$.

■

The sub-sampling algorithm as stated above seems to require a knowledge of the *exact* values of the timeout and the RTT. However, the same algorithm continues to work if we use any $I \geq \max\{\text{RTT, timeout}\}$.

5 Simulation Results

We built a simple implementation to test the feasibility and utility of our CFTP framework. We present results only for the case where each packet is lost independently with probability p. (Of course in this case the equilibrium distribution for the TCP Markov chain we have described could be calculated explicitly, but the number of states grows rapidly in the window size.)

Our implementation takes as input the maximum window size max_cwnd, and a drop probability p. The maximum slow start threshold is assumed to be the same as the maximum window size. We allow a congestion window of size one, and the minimum slow start threshold after a drop is two. Initially we simulate

Drop Rate p	max_cwnd= 40		max_cwnd= 400	
	Avg.	Max.	Avg.	Max.
0.001	1585.4	16384	5479.0	16384
0.01	350.8	1024	451.0	2048
0.02	174.1	1024	211.8	1024
0.03	121.3	512	130.9	512
0.04	83.6	256	93.2	512
0.05	68.7	256	72.7	512
0.06	53.6	256	56.7	256
0.07	44.3	256	47.6	256
0.08	38.4	256	44.3	256

Table 1: Quick samples from the CFTP method.

Drop Rate p	max_cwnd= 40	max_cwnd= 400
	Avg. CWND	Avg. CWND
0.001	33.71	47.36
0.01	14.16	14.25
0.02	9.77	10.09
0.04	6.93	7.07
0.08	4.97	5.02

Table 2: Average CWND value from CFTP samples.

two steps of the Markov chain, and then we double the number of time steps simulated if coupling has not occurred between the upper and lower bound states. Hence when we describe the number of steps required before coupling, our implementation always gives a power of two. Recall that this is not a requirement, but simply a convenient choice.

In Table 1, we show the average and maximum number of time steps (or equivalently packets) required before a state was output over 1,000 trials and various drop rates when the window size was set to 40 and 400 packets. The average is significantly smaller than the maximum; often CFTP yields an exact sample quickly. When the probability of a drop p is small, the coupling time is essentially dominated by the time for the congestion window of the lower bound state to reach its maximum size max_cwnd. When p is larger, the bounding states tend to couple more quickly, as the congestion window of the upper state quickly decreases toward the lower state. As predicted in the theory of Section 4, scaling up to a large maximum window size does not dramatically increase the running time required, particularly when the drop rate is high. Importantly, this suggests that CFTP may be a useful alternative approach when the number of states grows too large for explicit calculations of the equilibrium distribution.

In Table 2, we show the average CWND for our simulation. Note that this average is the average CWND experienced by each packet, and not the average over time. It is interesting to note that CWND still appears to follows the square root law (see e.g. [19]) in our simulations; that is, CWND falls roughly proportionally to $1/\sqrt{p}$. Also, as one might expect, the difference between the average CWND value does not differ significantly between the smaller and larger window size unless the drop rate is low.

We also provide results comparing our CFTP implementation to a *ns* simulation with 10,000 packets using the Tahoe protocol. We examined the specific case where the drop probability p is 0.01 and the maximum

congestion window, max_cwnd, is 40. We re-ran our CFTP simulation to obtain 1,000 new samples for this comparison. We found that the average congestion window over all packets was 14.77 in our CFTP samples, while it was 14.33 for the *ns* simulation. As another point of comparison, 5.2% of the samples were in the slow start mode for our CFTP samples, while 5.7% of the packets were in slow start mode for the *ns* simulation. We would expect the agreement to be rough, both because we are sampling and because our Markov chain is not a true faithful representation of the Tahoe protocol. These results suggest that our approach can lead to good approximations for actual TCP behavior.

6 Conclusions

We presented a partial order on the space of TCP window states that possesses a natural monotonic property. This leads to an efficient application of the "Coupling From The Past" paradigm to sample from the stationary distribution of TCP window states as seen by a random packet. The convergence time of this scheme is $O(N_{mix} \ln W)$ where N_{mix} is the number of steps required for the underlying TCP Markov chain to mix and W is the maximum window size of. It is unrealistic to expect an exact sample in less than N_{mix} steps. Hence, the above algorithm is only a small factor ($\ln W$) away from the optimum. At the same time, the algorithm does not need to know N_{mix}. We also showed how a simple sub-sampling algorithm can be used to obtain a sample at a random time instant. Our partial order and proof of monotonicity may well be of independent interest and may yield new insight into the structure of TCP congestion control algorithms. Our simulations of simple scenarios suggest that this approach is efficient, scales well with increasing maximum window sizes, and yields results which are close to those obtained by running the network simulator *ns* for TCP-Tahoe.

The above approach can potentially give rise to the following new paradigm for network simulations: instead of simulating a protocol over long times, or explicitly finding the stationary distribution of the states of the protocol, try to quickly obtain a "typical" sample of the state of the protocol. Our approach currently works for a simplified TCP model that does not employ fast recovery. Extending these ideas to other TCP variants and other network protocols is an important open problem. Also, it would be interesting to develop a simulation infrastructure to explore the practical utility of the ideas in this paper.

References

[1] D.J. Aldous. A random walk construction of uniform spanning trees and uniform labelled trees. *SIAM Journal on Discrete Mathematics*, 3(4):450–465, 1990.

[2] M. Allman, V. Paxson, and W. Stevens. RFC 2581: TCP Congestion Control. 1999.

[3] Eitan Altman, Konstantin Avrachenkov, and Chadi Barakat. A stochastic model of TCP/IP with stationary random losses. In *SIGCOMM*, pages 231–242, 2000.

[4] S. Asmussen, P. Glynn, and H. Thorisson. Stationary detection in the initial transient problem. *ACM trans. on modeling and comp. simulation*, 2(2):130–157, 1992.

[5] A. Broder. Generating random spanning trees. *30th Annual Symposium on Foundations of Computer Science*, pages 442–447, 1989.

[6] Neal Cardwell, Stefan Savage, and Thomas Anderson. Modeling TCP latency. In *INFOCOM*, pages 1742–1751, 2000.

[7] S. Cho and A. Goel. Exact sampling in machine scheduling problems. *To appear in the 5th international workshop on Randomization and Approximation Techniques in Computer Science (RANDOM '01)*, 2001.

[8] P. Diaconis and L. Saloff-Coste. What do we know about the metropolis algorithm? *Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 112–129, 1995.

[9] K Fall and S Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *Computer Communication Review*, 26(3):5–21, 1996.

[10] J. Fill. An interruptible algorithm for perfect sampling via Markov chains. *Annals of Applied Probability*, 8(1):131–162, 1998.

[11] M. Huber. Exact sampling and approximate counting techniques. *30th ACM Symposium on the Theory of Computing*, pages 31–40, 1998.

[12] V. Jacobson. Congestion avoidance and control. *Computer Communication Review*, 18(4):314–29, 1988.

[13] M. Jerrum and A. Sinclair. The Markov chain Monte Carlo method: an approach to approximate counting and integration. *In "Approximation Algorithms for NP-hard Problems," D.S.Hochbaum ed.*, 1996.

[14] L. Lovász and P. Winkler. Exact mixing in an unknown Markov chain. *Electronic Journal of Combinatorics, 2, paper #R15*, 1995.

[15] Vishal Misra, Wei-Bo Gong, and Donald F. Towsley. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *SIGCOMM*, pages 151–160, 2000.

[16] M. Mitzenmacher and R. Rajaraman. Towards more complete models of tcp latency and throughput. *Journal of Supercomputing*, 2001.

[17] ns: UCB/LBNL/VINT network simulator. *http://www-mash.cs.berkeley.edu/ns/*.

[18] J. Padhye, V. Firoiu, and D. Towsley. A stochastic model of TCP Reno congestion avoidance and control. *CMPSCI Technical Report 9902, University of Massachusetts, Amherst, MA*, 1999.

[19] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A simple model and its empirical validation. In *ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 303–314, Vancouver, CA, 1998.

[20] J.G. Propp and D.B. Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structure & Algorithms*, 9:223–252, 1996.

[21] W. R. Stevens. *TCP/IP Illustrated, Volume 3; TCP for Transactions, HTTP, NNTP and the UNIX Domain Protocols*. Addison Wesley, Reading, 1995.

[22] M. Zorzi, A. Chockalingam, and R. Rao. Throughput analysis of tcp on channels with memory. *IEEE J. Selected Areas Comm.*, 18:1289–1300, July 2000.