Formal Analysis of Passive Measurement Inference Techniques

Sharad Jaiswal[‡], Gianluca Iannaccone[§], Jim Kurose[†], Don Towsley[†]

[‡]Bell Labs India
Bangalore, India
jsharad@lucent.com

[§]Intel Research
Cambridge, UK
gianluca.iannaccone@intel.com

[†]Computer Science Department
Univ. of Massachusetts, Amherst
{kurose,towsley}@cs.umass.edu

***Abstract*— Verifying the accuracy of a passive measurements-based inference technique under all possible network scenarios is a difficult challenge - the measurement point has limited observability of events along the path, and monitored paths can exhibit a wide range of network properties (packet loss, reordering, end-end delay, route changes). In this paper, we propose and apply formal verification techniques to exhaustively verify the correctness of an inference technique. We apply this approach to the problem of inferring packet retransmissions and reorderings from passively observed packets at a single measurement point. We define classification rules for this inference problem and, through a combination of model-checking and formal reasoning, uncover all possible events in the network for which the rules produce incorrect inferences. Our work is novel in its use of formal verification tools for evaluating inference techniques in network measurements.**

I. Introduction

Inferring network properties through passive measurements is an interesting, challenging, and growing area of research. A monitor is placed on a network link and passively observes packets passing through the link. One can then apply inference techniques on these observations to compute performance metrics [3], [10], [14], [13], [11], study protocol behavior [26], [11] and understand the dynamics of traffic demands in the network.

The primary advantage of passive measurements is its scalability. From a single measurement point, a traffic monitor can observe a very large and diverse set of end-end paths, connections and end-hosts. However, the nature of the measurements also presents some challenging problems. Since the monitor can be located anywhere on the end-end path, it has limited knowledge of all the events that occur along the path. Also, since the monitored traffic might arise from and traverse a very heterogeneous set of sources and end-end paths, inference techniques must deal with a wide range of path properties (end-end delay, packet loss, reordering rates etc.) and network conditions along the path (link failures, route changes etc.).

The *limited observability* of network events and their *heterogeneity* underscore one of the main problems behind passive measurements: *"How to determine whether the inferences made by the passive monitor are correct and complete (i.e. all events are detected)?"* Simulations and real world experiments allow one to validate inference techniques but they fall short of emulating *all* possible network scenarios under which the techniques may fail.

The goal of this work is to introduce a new approach for the evaluation of passive measurement based inference techniques through an exhaustive formal analysis over all possible events in the network.

Towards this end, we revisit a well-known inference problem [3], [10], [26]: the identification of retransmitted and reordered packets within a packet stream as observed by a monitoring point in the middle of a TCP connection's end-end path. This problem plays an important role in network troubleshooting and diagnosis given that packet retransmissions are a measure of the end-end congestion and loss-rate in the path, while reorderings and replications may indicate the presence of faulty equipment, route flaps, or routing loops.

Our approach relies on *model checking* and reasoning about protocol and network behavior. The main challenge in model checking is dealing with the state space explosion problem. This problem occurs in systems with many components that can interact with each other or systems with data structures that can assume many different values. In our case, the problem surfaces because the communication protocol has input parameters that take a very large range of values (i.e., sequence numbers). We will use a model checker to discover whether the classification rules fail for some specific (and small) values of these parameters. Then, with these cases as a guide and by reasoning about protocol and network behavior, we prove properties of the classification rules for all possible values of the input parameters.

The contributions of this work can be summarized as follows. We consider classification rules for the detection and classification of packet retransmissions and reorderings by a passive measurement point. We verify the correctness of these rules using the SPIN model checker [8] with the end-points implementing the *Go-back-N* ($GB(N)$) protocol. For small values of N, we identify the set of cases for which the classification rules cannot detect packet retransmissions. By reasoning about the behavior of $GB(N)$ protocol, we then prove properties about the classification rules for any value of N. We then consider the case of the end-points implementing a TCP-like protocol, termed $FastRetx(N)$, and again verify the correctness of the classification rules.

This paper is organized as follows. We begin with a discussion of related work in Section 2. We precisely define our verification problem and give a high level road map of our approach in Section 3. Section 4 describes in detail how we implement the different components of the system and carry out the verification in SPIN. In Section 5, we introduce the $GB(N)$ protocol, and verify our inference rules using SPIN, for the cases $N = 2, 3, 4$. We then provide a formal proof in Section 6, to extend these results for any value of N. Then, in Section 7, we look at a more complicated protocol that we term $FastRetx(n)$ that has TCP-like loss recovery mechanisms, and verify the classification rules for this protocol. We then take a step back and discuss the implications of our observations for the different flavors of TCP in Section 8. Finally, we conclude with a discussion of future directions in Section 9.

II. Related work

There exists a significant body of literature about the formal verification of communication protocols. Also, the technique of model checking has also been extensively used to verify network protocols. In [20] and [19] the authors use the model checker CMC to identify bugs in the Linux TCP implementation and the AODV routing protocol. The SPIN model checker, in particular, has been widely used, e.g. in [2][6] to verify protocol properties. For a further extended survey of formal verification applications using SPIN, the reader is referred to [8] and proceedings of the SPIN workshops [1]. There also exist other works in the literature that, similar to our approach, use a combination of model checking and formal reasoning for protocol verification, e.g. [22], [23] take a formal reasoning approach to prove properties about Transaction-TCP and TCP-SACK protocols, and [5] uses a combination of SPIN and formal reasoning to verify properties of distance vector routing protocols. Our work is the first to use the tools of formal verification towards the evaluation of measurement techniques. Verification of a measurement technique has to take into account the interactions between the protocol sender and receiver behavior, the channel behavior and the inference technique itself. In subsequent sections we describe the framework using which we verify the classification rules at a passive measurement point.

III. Problem definition and approach

Our inference problem is as follows. Consider a data sender and a receiver communicating through a protocol that ensures reliable in-order delivery of messages. Messages can be dropped and reordered in the end-end path. A passive monitor is located in the path, and observes all packets between the sender and the receiver. Figure 1 illustrates this scenario. The measurement point implements inference rules to identify all packet retransmissions and reorderings in the connection between the sender and the receiver. Our goal is to verify, whether these rules can *correctly* identify *all* retransmitted and reordered packets that appear in the end-end path.

We model the various components of the network, namely the protocol behavior at the sender and receiver, the communication channels (that induce packet loss and reordering) and the measurement point's inference rules, using a framework similar to the *Communicating Finite State Machine* framework (CFSM) that has previously been used [18] for protocol specification and for modelling a network. A CFSM models the nodes of a network as a set of finite state machines, and a set of channels that represent the communication channels between any two nodes. Thus, it is a natural and simple formalism to specify our network model.

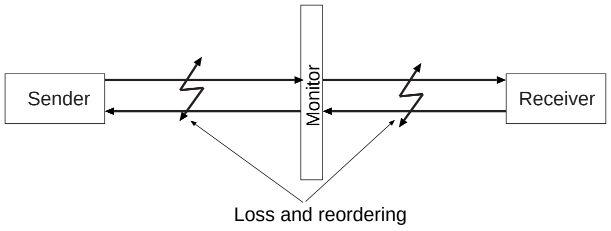

Fig. 1. The network scenario

As a first step we implement this model in the SPIN model checker [8]. The SPIN model checking system has been widely used to verify communication protocols. The protocol and the inference heuristics are specified using the Promela language and a protocol simulator can perform random or guided simulations. The model checker performs an exhaustive state-space search to verify that a given property holds under all possible simulations of the system.

In our case, the model checker generates all possible packet loss and reordering events in the channels. It then checks which (if any) retransmissions and reorderings are not detected by the measurement point's inference rules.

Model checking has a number of advantages over traditional approaches based on simulation, testing, and deductive reasoning. In particular, model checking is automatic and usually quite fast. Also, if the design contains an error, model checking will produce a counterexample that can be used to pinpoint the source of the error.

However, as discussed earlier, the model checker is limited by the state explosion problem. Hence it can handle only some specific and simple values of the protocol's input parameters. Using the cases generated by the model checker for which the inference rules are found to be incorrect, we extend the results for all values of the protocol's input parameters through formal reasoning about the protocol behavior.

IV. Specification and Verification in SPIN

In this section, we present a detailed specification of the state machines of the system and the steps leading to the verification in SPIN. The system consists of the following components:

Sender and Receiver. These two state machines (M_s and M_r) implement the transport protocol that governs the communication between the end-hosts. In [12] we provide the Promela code for the various transport protocols that we investigate.

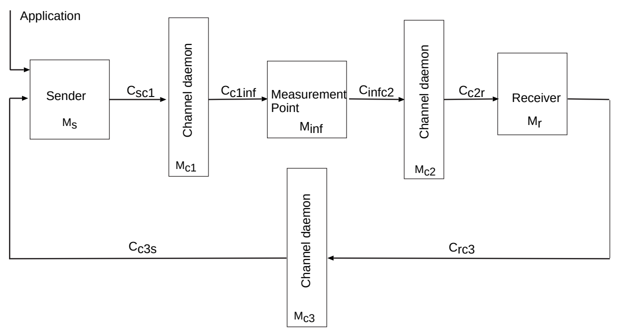

Fig. 2. The network model

Channel daemons. M_{c1}, M_{c2}, M_{c3} are the representations of channel daemons between the sender and the measurement point, the measurement point and the receiver, and the receiver and the sender respectively. These daemons create loss and reordering events in the communication channels between the two end points. As described in the state machine description in [12], these daemons snoop on the channel, and if there is a packet present, they take one of three actions: (i) let the packet go through with no interference; (ii) remove and drop the packet from the channel; (iii) remove the packet, change its ordering with the subsequent packet, and put them back in the channel.

Measurement point. M_{inf} represents the measurement point daemon. This daemon is a passive observer of packets passing between the protocol sender and receiver, and encodes the rules for detecting and distinguishing between retransmissions and reorderings. The classification rules assume certain properties about protocol behavior, namely that the packets carry monotonically increasing sequence numbers, and that transmission is reliable, i.e. lost packets are retransmitted until received successfully. The inference rules also assume that the timescale at which reordering events occur in the network is smaller than the round trip propagation delay of the connection.

Any data packet observed at the measurement point is referred to by the notation (x, t), where x is the sequence number of the packet, and t is the time at which the packet was observed. A packet is defined to be *out-of-sequence* if it has a sequence number x less than or equal to that of other previously observed packet.

When the measurement point does observe an out-of-sequence packet it classifies it as a retransmission or a reordering based on the following rules

- **classify.retx:** Let (x, t_{orig}) refer to the packet with sequence number x sent by the sender, and observed at time t_{orig}. If after some time, we observe an out-of-sequence packet (x, t_{oos}), with the same sequence number x and $t_{oos} > t_{orig}$, then (x, t_{oos}) is a *retransmission*.
- **classify.reord:** Suppose the measurement point observes an out-of-sequence packet (x, t_{oos}), and let t_{inseq} ($t_{inseq} < t_{oos}$) be the earliest time at which a packet with a sequence number greater than x is observed, we then look at the *timelag* $t_{oos} - t_{inseq}$,
  - if $t_{oos} - t_{inseq} >= RTO$, (x, t_{oos}) is a *retransmission*
  - if $t_{oos} - t_{inseq} < RTT$, (x, t_{oos}) is a *reordering*

Here RTO is the value of the sender's retransmission timer (that we assume the measurement point knows) and RTT is the measurement point's estimate of the round trip delay of the end-end path.

We also introduce C_{sc1}, C_{c1inf} , C_{infc2}, C_{c2r}, C_{rc3} and C_{c3s}, as the channels between the various components. A representation of the system is illustrated in Figure 2.

We must now choose the communication protocol used by the sender and the receiver. As a first step, we study the inference techniques with the end points implementing the *Go-back-N* ($GB(N)$) protocol. We choose $GB(N)$ because it is the simplest window-based protocol, and a tractable first step in analyzing more complicated window-based protocols such as TCP. Later, we extend our analysis to a more complicated protocol, which while retaining features of $GB(N)$ such as a fixed window, and a go-back-N timeout mechanism, also incorporates TCP like mechanisms such as loss recovery through fast retransmit. Details of the two protocols will be presented in subsequent sections.

Our goal is to determine whether all retransmitted and reordered packets that reach the measurement point are detected as out-of-sequence by the measurement point, and then classified correctly to be either a retransmission or a reordering. In order to perform this check with SPIN, we introduce a *type* field in the header of any packet transmitted by the sender. This field can take one of three values: `<Norm, Retx, Reord>`. When a packet is transmitted by the sender for the first time, its type field is set to `Norm`. If this packet is dropped by the channel daemon process, then when a packet with the same sequence number is retransmitted by the sender, its type is set to `Retx`. Similarly, if the channel daemon reorders the position of this packet with a packet sent later in time by the sender, it rewrites the type field to `Reord`.

The model checker can generate all network events that would cause packet retransmissions and reorderings. When the measurement point observes a packet, it checks if this packet is out-of-sequence based on whether the packet has a sequence number less than or equal to that of a previously observed packet. If this is the case, the measurement point invokes the rules *classify.retx* and *classify.reord* and classifies the packet to be a retransmission or a reordering. Finally, it compares its decision with the type field in the packet, and if the two do not match, we have a case in which a packet was misclassified by the measurement point.

A. Discussion of model assumptions

Until now, in this section we have described our system model in detail. As stated, our primary goal is to verify whether the measurement point can detect and correctly classify all packet retransmissions and reorderings. We have made several assumptions in our model to simplify the verification process. We now discuss, how (and if) each of these assumptions impacts the completeness of our study. Firstly, we have excluded the occurrence of packet replications in the end-end path. While our intention is to allow all possible events that result in out-of-sequence packets in order to comprehensively test the classification rules, we exclude packet replications since it makes verification simpler, and several measurement studies [9], [21] have indicated that packet replications are an extremely rare phenomenon in the Internet.

Our classification rules determine whether an out-of-sequence packet is a retransmission or a reordered packet by comparing its *timelag* with the connection's RTT and RTO. We assume that the measurement always has an accurate estimate of the connection's RTT and RTO (the reader is referred to [11], [25] for techniques that provide estimates of e.g., a TCP connection's RTT/RTO from a passive measurement point). We also impose restrictions on the time scale of packet reordering and retransmission events. It is clear that our classification rules may give incorrect results if the measurement point has an incorrect estimate of the sender's RTT and RTO, or if the time scales of these events do not conform to our assumptions.

However, our goal in this verification exercise is not to verify the assumptions around which the classification rules were designed. Instead we aim to investigate whether there exist network scenarios that would cause the classification rules to give incorrect results even when all the assumptions hold. As with all verification efforts, the results of the verification process hold only when the stated assumptions hold true.

Finally, we restrict reordering events to occur only between successive packets. We make this simplifying assumption since the only aspect of packet reordering that determines the decision of the classification rules is the *timelag* of the reordered packe. Thus, whether a packet is reordered with a subsequent packet, or across multiple packets, are equivalent events as long as the *timelag* of the reordered packet, satisfies our assumptions about the time scale of reordering events.

V. Verifying inference heuristics for $GB(N)$

In this section we consider the $GB(N)$ protocol and prove its correctness even in the case of in-network reordering of packets. We then show the correctness of the measurement point's out-of-sequence inference technique using SPIN.

A. Correctness of GB(N) in the presence of reordering

The $GB(N)$ protocol, in brief, operates as follows. The sender can have as many as N packets outstanding in the channel at any point of time. The receiver has a buffer of one, and hence only accepts packets that are in sequence. When a packet is dropped, the sender recovers from loss only by using a timeout mechanism. If no acknowledgement is received before the timeout expires, the sender retransmits all packets that are yet unacknowledged. More details of the $GB(N)$ protocol can be found in [24], [15], and a description of the $GB(N)$ sender and receiver state machines are presented in [12].

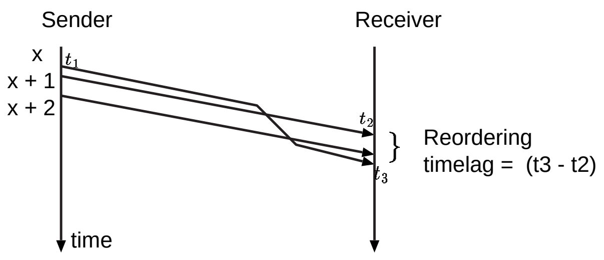

Fig. 3. Timelag of a reordering event: packet with sequence number x is reordered with subsequently sent sequence number $x + 1$

$GB(N)$ uses a sequence number variable carried by both data and ACK packets. The range of values taken by this variable impacts the model checking process – a large range leads to a state space explosion when the model checker performs an exhaustive state space search. In order to get around this problem, we use a version of the $GB(N)$ protocol that uses a small bounded range of sequence numbers. However, this requires us to ensure that the protocol behaves correctly, in the presence of packet loss and reordering, despite this constraint. The protocol behaves correctly if the receiver never releases a packet out of the correct order to the higher layer. More formally, let $\{p_1, p_2, ..., p_n\}$ be packets released by the sender with their indexes indicating the order in which they are sent and let $\{x_1, x_2, ..., x_n\}$ be their respective sequence numbers. A protocol behaves correctly if the receiver (by examining the sequence numbers of the received packets) always sends these packets up to the higher layer in the order in which they are sent.

It has been earlier noted [4], that the $GB(N)$ protocol behaves correctly if the protocol adopts a sequence number range which is equal to m, where $m > N$. However the proof of this property makes the assumption that packets are delivered *in sequence* by the communication channel between the sender and the receiver. This assumption is not true in our context since the channel daemons can reorder packets along the end-end path. We now establish conditions under which $GB(N)$ protocol behaves correctly with a bounded range of sequence numbers in the presence of reordering in the end-end path.

We first define the notion of *timelag* of a reordering event. Let a packet with sequence number x be released by the sender at time t_1. Let $x + 1, x + 2$ be the sequence numbers of packets released by the sender at a point in time later than t_1. Now suppose packet x is reordered, i.e., it arrives at the receiver later than some packet with sequence number $> x$. Also let t_2 be the time at which the first of any packets with sequence number $> x$, reaches the receiver, and t_3 the time when x finally arrives at the receiver. Then the timelag of this reordering event is the difference $t_3 - t_2$. This is illustrated in Figure 3.

Theorem 1: The correctness of the $GB(N)$ protocol is preserved if the sender uses a bounded range of sequence numbers m, where $m > 2N$, and the timelag of any reordering event is always less than the minimum two-way propagation delay D.

Proof: Suppose a packet, p, with sequence number (S mod m) is released by the sender and reordered in the channel (i.e., it is exchanged with a packet sent later in time by the sender). Now suppose the maximum sequence number observed at the receiver before packet p eventually reaches the receiver is (($S+x$) mod m). First, consider the case when $x < 2N$. Since $m > 2N$, it is easy to see packet p could never be erroneously accepted by the receiver, since there could not have been any sequence number wrap around between (S mod m) and (($S+x$) mod m).

Now consider the case when $x \geq 2N$ i.e., (($S+2N$) mod m) has been observed at the receiver. This would imply that the packet with sequence number (($S+N$) mod m) has also been observed by the receiver, since sequence number (($S+2N$) mod m) could have been sent only after the sender received an ACK for (($S+N$) mod m). Now, the interval between when sequence numbers (($S+2N$) mod m) and (($S+N$) mod m) arrive at the receiver must be $\geq D$. This implies that the reordering timelag of packet p is also greater than D, since the packet with sequence number (($S+N$) mod m) was transmitted later than (S mod m). However, we have imposed a restriction on the channel that no reordering event occurs with a timelag larger than D. Thus when a reordered packet with sequence number (S mod m) arrives at the receiver, the maximum sequence number observed must be less than (($S+2N$) mod m), and so the protocol functions correctly.

$\square$

B. Detecting retransmitted and reordered packets

As suggested by Theorem 1, we adopt $2N+1$ as the range of sequence numbers used by the $GB(N)$ specification in SPIN, and, to ensure the correctness of the protocol, impose the constraints that reordering timelags are less than the round trip propagation delay of the connection. Apart from these restrictions to guarantee the correctness of $GB(N)$, we make the following additional assumptions about network and sender behavior throughout the rest of the paper:

- *The channel daemons do not cause any packet replications.* A replicated packet would also manifest itself as an out-of-sequence packet at the measurement point and must be distinguished from retransmissions and reorderings for correct classification. However, measurement studies [10] have shown that packet replications are extremely rare in the Internet, and hence we exclude their occurrence from the model.
- *The channel daemon between the measurement point and the receiver does not reorder any packets.* A packet reordered after the measurement point will not be out-of-sequence at the measurement point, hence the measurement point cannot detect this reordering event, and it is also excluded from the model.
- *The sender always has data to send, and the retransmission timer of the sender is not triggered before it has sent all packets allowed by its window.* We make these assumptions to exclude trivial cases in which the measurement point cannot detect packet retransmissions, such as if the last packet of the session is lost before the measurement point and is then retransmitted, or if a packet is lost, and retransmitted before the sender could transmit any more packets. Removal of these cases also simplifies the verification by SPIN.

Furthermore, we initially consider the case where there are no losses or reorderings of an ACK packet and that a packet is never retransmitted because of a delay in the arrival of its ACK (we refer to this case as *unneeded retransmissions*). These last two assumptions allow us to significantly simplify the verification in SPIN but later we will relax them and verify the correctness of the classification rules when ACKs can be lost or delayed.

Our goal is to verify the following two properties of the classification rules using the model checker with the $GB(N)$ protocol:

1) the *completeness* of the inference heuristics, i.e., whether all retransmitted and reordered packets that reach the measurement point are identified as out-of-sequence packets.
2) the *correctness* of the classification rules, *classify.retx* and *classify.reord* i.e., the measurement point never misclassifies an out-of-sequence packet, given the assumptions we make about the sender and the network behavior.

We used the model checker for a $GB(N)$ protocol with $N \leq 4$ (see [12] for the Promela code used in this work). The SPIN model checker identified the following cases under which the measurement point could not detect an out-of-sequence packet.

Let there be a packet with sequence number x, transmitted by the sender at time t' and lost *before* the measurement point. A subsequent *retransmission* with the same sequence number x transmitted by the sender at time t'', is not detected to be out-of-sequence at the measurement point when any of the events $\{E1(x,t''), E2(x,t''), E3(x,t'')\}$ occur (see Figure 4):

Event $E1(x,t'')$: The packet with sequence number $x-N+1$ was also lost before the initial transmission of sequence number x was lost.

Event $E2(x,t'')$: The packet with sequence number $x-N+1$ was reordered in sequence with an earlier sent packet.

Both $E1(x,t'')$ and $E2(x,t'')$ involve a packet with sequence number x lost before the measurement point, and the packet with sequence number $x-N+1$ dropped (or reordered) in the path. As mentioned earlier, a $GB(N)$ receiver only accepts those packets which are in order. If the packet with sequence number $x-N+1$ is lost ($E1$), none of the subsequent packets that reach the receiver are acknowledged. Also, if $x-N+1$ does not arrive in order ($E2$), it is dropped by the receiver,

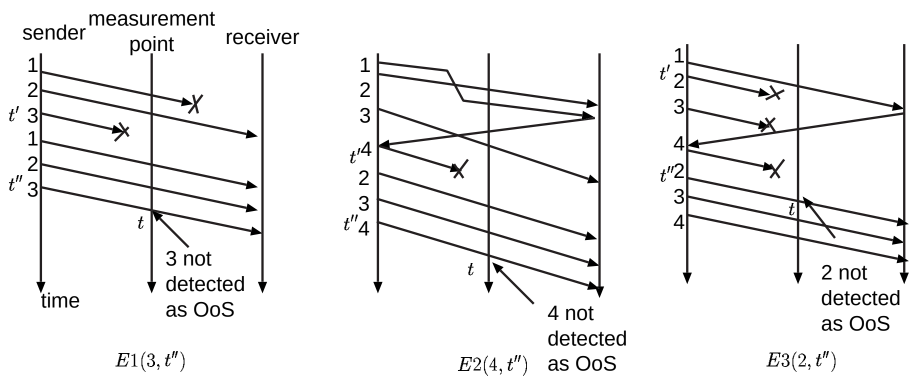

Fig. 4. Cases in which retransmitted and reordered packets are undetected at measurement point, for $GB(3)$, $m = 7$.

and this has the same impact as when $x - N + 1$ was lost in the path.

At this point, the sender has N unacknowledged packets in the channel (sequence numbers $x-N+1$,...,x), and the receiver will not send any new acknowledgement forcing the sender to wait for the timeout to expire. Thus, the retransmission of packet x will not be detected as an out-of-sequence packet by the measurement point given that no packet with a sequence number larger than x has ever been sent.

Event $E3(x, t'')$: All of the possible packets with sequence numbers $y \in \{x + 1, ..., x + N - 1\}$ sent by the sender between the first transmission at t' and t'' were also dropped.

Event $E3(x, t'')$ describes the case when a packet with sequence number x is lost before the measurement point, and all subsequent packets sent with sequence numbers $> x$ are also dropped before the measurement point. Again, it is easy to see that when the retransmission of sequence number x arrives at the monitor, it will not be considered to be out-of-sequence, as shown in Figure 4.

C. ACK losses

So far, we have not considered the case of acknowledgement loss by the channel. We now relax this assumption by adding a channel daemon between the receiver and the sender. Upon the arrival of an ACK, the daemon either lets the packet through, or drops it. With this addition, we again submit our model to SPIN. SPIN now exposes a new event for which the measurement point cannot detect a retransmitted packet as being out-of-sequence (Figure 5):

Event $E4(x, t'')$: A packet with sequence number x, transmitted by the sender at time t' is lost *before* the measurement point. For all packets with sequence numbers $x - N + 1, ..., x - 1$, sent prior to t', one of the following events occur:

- the packet is lost in the path.
- the packet reaches the receiver and if it generates an ACK that cumulatively acknowledges sequence number $x - N + 1$, this ACK is lost.

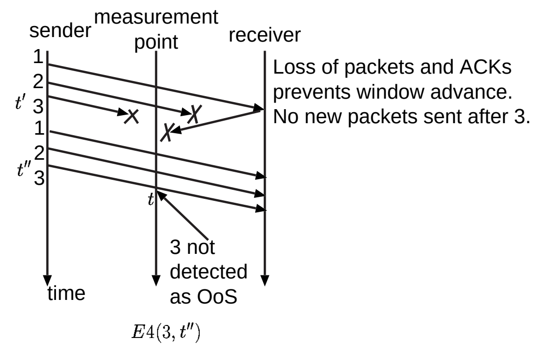
sender
measurement point
receiver
Loss of packets and ACKs prevents window advance. No new packets sent after 3.
time
3 not detected as OoS
$E4(3,t'')$

Fig. 5. Case $E4(x, t'')$, $GB(3)$

A subsequent retransmission of x sent at time t'' is not detected to be out-of-sequence by the measurement point.

The impact of lost ACKs is to prevent the sender's window from freeing up and can thus restrict the transmission of new packets. For the sender's window to move beyond a sequence number x, it should receive at least one new ACK for the packets, with sequence numbers $\{x-N+1, x-N+2, ..., x-1\}$, sent prior to an earlier version of the packet with sequence number x. We have already shown that this cannot occur if events $E1(x, t'')$ and $E2(x, t'')$ occur. However, even if these two events do not occur, but all the $N - 1$ packets sent prior to sequence number x are either lost, or if they generate an ACK that acknowledges sequence number $x - N + 1$, and this ACK is lost, then its easy to see that no new ACKs will reach the sender after the initial transmission of x, and its window will not move beyond sequence number x.

D. Unneeded Retransmissions

We now remove the assumption that acknowledgements are never delayed enough to force the sender to retransmit a packet that has already reached the receiver. Thus we now allow for the sender to retransmit a packet while the initial transmission of that sequence number (and its corresponding ACK) are still in the channel. In order to verify this modified network model we need to assume that there are no losses on the

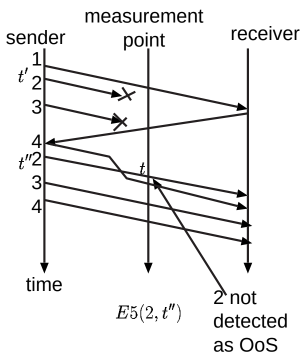

Fig. 6. Case $E5(x,t'')$, $GB(3)$: Packet with sequence number $x = 2$ is lost before the measurement point. Packet with sequence number $y = 4$ is the first packet with sequence number > 2 sent by the sender that reaches the measurement point. The retransmission of 2 is reordered with sequence number 4 and is not detected as OoS at the measurement point.

acknowledgement path given that SPIN could not complete the verification (due to state-explosion) even for small values of N.

We then run the model checker for $GB(N)$ with $N = \{2, 3\}$ and identify a new event in which a retransmitted packet is not detected by the measurement point (see Figure 6).

Event $E5(x,t'')$: Let a packet with sequence number x, transmitted by the sender at time t', that is lost *before* the measurement point. Let y be the sequence number of the first packet sent by the sender after t', such that $y > x$. Consider a subsequent *retransmission* with the same sequence number x transmitted by the sender at time t''. This retransmitted packet is not detected to be out-of-sequence if it is reordered with sequence number y before the measurement point.

As clearly illustrated in Figure 6, the packet with sequence number $x = 2$ will be mistaken by the measurement point for the original transmission and be inevitably not deemed out-of-sequence.

E. Classifying out-of-sequence packets

We now proceed to the second part of our verification which is to show that when a measurement point observes an out-of-sequence packet, the decision it makes (whether the packet is a retransmission or a reordering) is consistent with what actually happened in the network. Now, as we have described in the earlier section, the classification rules rely on the different timelags of packet retransmissions and reorderings to distinguish between the two (see Section IV).

The measurement point makes its decisions based on the time interval between when the measurement point observes an out-of-sequence packet x, and when it observes packet x', where x' is the first packet with a sequence number greater than x that is observed at the measurement point. The magnitude of this interval determines whether the OoS packet is a reordering or a retransmission.

However, the model checker has no built-in notion of time and there is no mechanism to quantify the time between any ordered sequence of events. Thus, in order to get around this problem, we impose the property that when reordering or retransmission occurs, the timelags conform to the assumptions. As a result of this assumption, the model checker throws up only one case in which the measurement point's decision does not match the type field carried by the packet. This happens when a retransmitted packet is itself reordered with other packets. In this case the resulting out-of-sequence packet is simply classified to be a retransmission, since the timelag of this packet is still greater than the RTO of the sender. Note that in this case the packet would belong at the same time to two states (retranmitted and reordered) that the rules in Section IV explicitly do not allow.

VI. Generalization to any N

The events described so far were discovered by SPIN for small values of N. The next question is whether these are the only events that result in a retransmitted or reordered packet going undetected at the measurement point for *any* value of N of the $GB(N)$ protocol. Reasoning on the behavior of the protocol and our network model, we can prove that those events are the only ones that would cause a retransmitted or reordered packet to pass undetected by the measurement point.

Before proceeding, let us define some notation:

- $A(x,t,t'')$ is $true$ if some packet with sequence number x sent by the sender at time t'' arrives at the measurement point at a time t, and this packet is either a *retransmission* or a *reordering*.
- $P(x,t,t'')$ is true only if at some point in time before t, the measurement point also observed a packet with a sequence number x', such that $x \leq x'$, i.e., it is true if x is recognized as out-of-sequence by the measurement point.
- $Q(x,t'')$ is equal to $\neg(E1(x,t'') \vee E2(x,t'') \vee E3(x,t'') \vee E4(x,t'') \vee E5(x,t''))$, i.e., it is true if none of the events $E1 - E5$ has occurred.

We can then state the following.

Theorem 2: Suppose a *retransmitted* or *reordered* packet sent with sequence number x at time t'' using the $GB(N)$ protocol, for any N, arrives at the measurement point at time t. Also suppose that $Q(x,t'')$ is true. Then, the property $P(x,t,t'')$ will hold true, that is,

$$A(x,t,t'') \wedge Q(x,t'') \rightarrow P(x,t,t'') \tag{1}$$

Proof: Our proof is by contradiction. We initially assume that:

1) $A(x,t,t'')$ is true, i.e. a retransmitted or reordered packet reached the measurement point at time t. We shall refer to this packet as p.
2) $P(x,t,t'')$ is *not* true, i.e., the measurement point does not consider packet p out-of-sequence.

3) $Q(x, t'')$ is true, i.e., *none* of the events $E1 - E5$ have occurred.

We will show that if 1) and 2) hold, then 3) cannot be true, and one of the events $E1 - E5$ must have occurred.

If $P(x, t, t'')$ is false, the measurement point has not seen any packet with sequence number x' such that $x \leq x'$. This implies that p has to be a retransmission, since if it had been a reordered packet then a packet with a sequence number greater than x must have appeared at the measurement point before t. Now, the only ways in which the measurement point could not observe a packet with sequence number greater than or equal to x between the original transmission of a packet with sequence number x and p could be because:

- The sender could *not send* any packet with a sequence number x' greater than x since the original transmission of sequence number x. Since we assume the sender always has data to send, this can happen only because the sender has not seen an acknowledgment for any of the $N - 1$ packets sent previous to x. In order for this to occur, one of the following events *must* have happened:
  - the packet with sequence number $x'' = x - N + 1$ was lost in the channel.
  - the packet with sequence number $x'' = x - N + 1$ was reordered with an earlier sent packet in the channel.
  - or for all packets with sequence numbers $\{x - N + 1, x - N + 2, ..., x - 1\}$, either the packet is lost, or if it triggers an ACK that acknowledges sequence number $x - N + 1$, then this ACK is lost.
    This would basically result in the sender having N unacknowledged packets in the network and stopping it from sending more packets.

  Any of these events would freeze the advancement of the right edge of the sender window beyond x (as is illustrated in Figure 4), and no new packets with sequence number $> x$ would be transmitted. However these events correspond to $E1(x, t'')$, $E2(x, t'')$ and $E4(x, t'')$ which we assume to have not occurred.
- The sender could potentially send packets with sequence numbers $x+1, x+2, ..., x+N-1$ before it retransmitted x, and these were *all lost* before the measurement point could observe them. This corresponds to event $E3(x, t'')$.
- One of the packets with sequence numbers $x + 1, x + 2, ..., x + N - 1$, that the sender could potentially send before retransmitting x, made it to the measurement point. Let p' be the first of these packets. If p' is reordered in position with p by the channel daemon then the retransmitted packet p would not be considered to be OoS by the measurement point. This is event $E5(x, t'')$.

Hence, we have shown that with $GB(N)$ a retransmitted or reordered packet is not detected at the measurement point only if one of the events $E1 - E5$ occurs for any value of N.

□

To summarize, using SPIN together with formal reasoning we have uncovered all cases in which the measurement point can not detect packet retransmissions and reorderings as out-of-sequence packets.

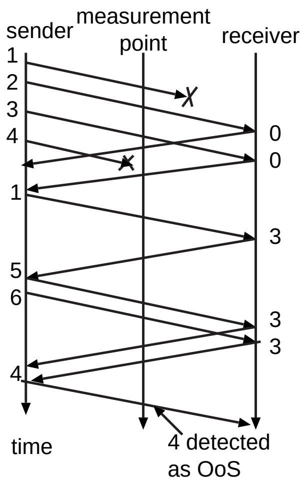

Fig. 7. Recovery through Fast Retransmit in $FastRetx(4)$, $dupacks = 2$

VII. Extensions to $GB(N)$

We now consider extensions to $GB(N)$ that more closely capture the behavior of TCP. We name this extended protocol $FastRetx(N)$. Just as in $GB(N)$, the $FastRetx(N)$ sender has a fixed window, and can have N packets outstanding at any point of time. However, there are two crucial differences. First, the $FastRetx(N)$ receiver has a buffer of size N, and also accepts and stores packets that do not arrive in order. The receipt of these packets triggers an acknowledgement that *cumulatively* specifies the packets received in order at the receiver. The second difference lies in how the sender detects and recovers from packet loss. Say a packet is dropped by the channel. As subsequent packets arrive at the receiver, they are buffered and trigger *duplicate* ACKs for the lost packet. When the sender receives a certain threshold (referred as the *dupack* threshold) worth of such duplicate ACKs, it goes ahead and retransmits the packet indicated to be lost by these duplicate ACKs. If this retransmission subsequently arrives at the receiver, it responds with a new ACK for the maximum sequence number it has received in sequence, and the sender continues with its normal transmission. In some cases however, if several packets are lost within a window, then the required number of duplicate ACKs may not be generated by the receiver or the ACKs maybe lost before reaching the sender. In this case, the sender times out and, just as in $GB(N)$, retransmits the entire window of packets starting from the first unacknowledged packet. A Promela implementation of $FastRetx(N)$ can be found in [12].

We have implemented a model of the $FastRetx(N)$ protocol in SPIN. We verified this model for $N = 4$ and the duplicate ACK threshold, $dupack = 2$. In our network model we allow losses before and after the measurement point, reorderings before the measurement point and ACK losses between the receiver and the sender. We examined the cases

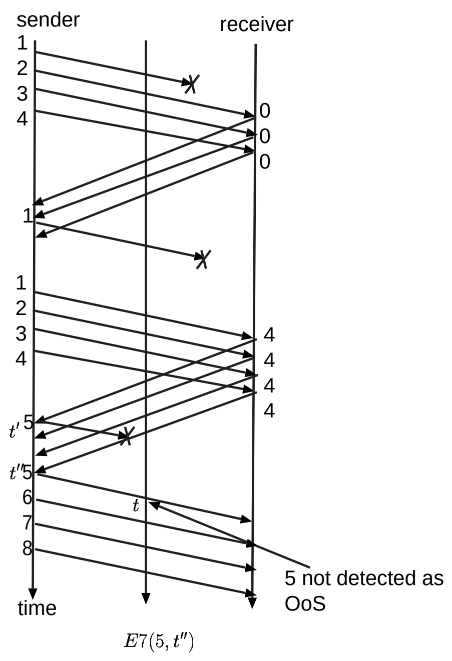

Fig. 8. Event $E7(x, t'')$. Packet with sequence number 5 is lost before measurement point and, due to a *successive fast retransmit* event, is retransmitted before sender could send any packets with sequence number > 5. The retransmission of 5 is not detected to be OoS.

under which retransmissions and reorderings are not detected as out-of-sequence packets at the measurement point, and our observations are summarized below:

1) SPIN again uncovered scenarios equivalent to events $E3(x, t')$,$E4(x, t')$ and $E5(x, t')$ (described for the $GB(N)$ protocol), that result in a retransmitted packet to be not detected as out-of-sequence at the measurement point.
2) In $GB(N)$, for any packet with sequence number x, if the first packet in the window with respect to this packet (namely the sequence number $x - N + 1$) is lost or reordered, then the sender's window does not move beyond sequence number x. Thus, if the packet with sequence number x is lost before the measurement point then its subsequent retransmission is not detected to be out-of-sequence - this constitutes events $E1(x, t')$ and $E2(x, t')$ for the $GB(N)$ protocol. In the $FastRetx(N)$ protocol, since an out-of-order packet is buffered and eventually acknowledged, we observe that the reordering of sequence number $x - N + 1$ does not impact the advancement of the sender's window beyond x. Hence, there is no event equivalent to $E2(x, t'')$ in $FastRetx(N)$. Moreover, we found that even if the packet with sequence number $x - N + 1$ is dropped, if the loss of this packet is detected through the fast retransmit mechanism, the sender's window *can* move beyond sequence number x. We explain this with an example illustrated in Figure 7. As shown in the figure, the packet with sequence number 1 is dropped, and also sequence number 4 is lost before the measurement point. When sequence numbers 2 and 3 arrive at the receiver, they trigger duplicate ACKs for sequence number 1. This results in a fast retransmit of sequence number 1, and subsequently a new ACK for 4. The sender now responds with two *new* packets with sequence numbers 5 and 6 (i.e. its window has now advanced beyond 4). When these packets are received, since 4 has still not been received, they again trigger duplicate ACKS, and sequence number 4 is finally retransmitted. However, 4 is now detected to be out-of-sequence (because the measurement point has seen sequence numbers > 4), unlike in $GB(N)$.
To summarize, suppose a packet (x, t') is lost before the measurement point, and also the packet with sequence number $x - N + 1$ is dropped. Now, *only if* sequence number $x - N + 1$ is not successfully retransmitted (and acknowledged) using the fast retransmit mechanism, will the sender's window fail to move beyond sequence number x. And when this happens, the retransmission of x will not be detected as out-of-sequence at the measurement point. We shall refer to this event as $E6(x, t')$.
3) SPIN uncovered a *new* event in which a retransmitted packet is not detected by the measurement point. This is a consequence of the *successive fast retransmit* event [7] that occurs as a result of the receiver sending multiple duplicate ACKs upon receiving packets that it has already buffered. As illustrated in Figure 8, sequence number 1 is lost, and retransmitted by the sender after receiving duplicate ACKs generated by the arrival of sequence numbers 2, 3 and 4 at the receiver. However, the retransmitted packet 1 is dropped again, triggering a timeout event and subsequent retransmission of all sequence numbers $1 - 4$ by the sender. Upon receiving these packets, the receiver responds with an ACK for sequence number 5 (since it has already received and buffered sequence numbers $2 - 4$). The sender transmits 5 after receiving the first of these new ACKs, however, this packet is dropped before the measurement point. The following three ACKs for 5 are then interpreted as duplicate ACKs by the sender and it again retransmits 5. This retransmission is not detected to be out-of-sequence by the measurement point, since the sender did not send any new packets after the initial transmission of 5. We refer to this event as $E7(x, t'')$.

We observed these events for $N = 4$, for the $FastRetx(N)$ protocol. We now show that these are the only events that result in a retransmission or a reordered packet to be not identified as out-of-sequence at the measurement point, for any value of N.

Theorem 3: Suppose a *retransmitted* or *reordered* packet sent with sequence number x at time t'' using the $FastRetx(N)$ protocol, for any N, arrives at measurement point at time t. Also suppose that none of the events $\{E3(x, t''), E4(x, t''), E5(x, t''), E6(x, t''), E7(x, t'')\}$ have occurred, then the property $P(x, t, t'')$ will hold true.

Proof: Once again, our proof is by contradiction, i.e. we initially assume that none of the events $\{E3(x,t''), E4(x,t''), E5(x,t''), E6(x,t''), E7(x,t'')\}$ has occurred, and that a retransmission or a reordered packet has arrived at the measurement, but $P(x,t,t'')$ is false. This leads to a contradiction, hence one of the events in the above described set must have occurred.

If $P(x,t,t'')$ is false, the measurement point has not seen any packet with sequence number x' such that $x \leq x'$. This implies that p has to be a retransmission, since if it had been a reordered packet then a packet with a sequence number greater than x must have appeared at the measurement point before t. Now, the only ways in which the measurement point could not observe a packet with sequence number greater or equal to x between the original transmission of a packet with sequence number x and this retransmission p could be due to one of the following:

- The sender could *not send* any packets with a sequence number x' greater than x since the original transmission of sequence number x. Since we assume the sender has always data to send, this can happen only if the sender has not seen an acknowledgment for any of the $N-1$ packets sent previous to x. In order for this to occur, either of the following events *must* have occurred:
  - the first packet of the window with respect to sequence number x, i.e. the packet with sequence number $x'' = x - N + 1$ was lost in the channel, *and* the sender could not recover from the loss using the fast retransmit mechanism. Since, if the sequence number $x - N + 1$ had been successfully retransmitted using the fast retransmit mechanism, the sender's window would have moved beyond x, before retransmitting x.
  - for all packets with sequence numbers $\{x - N + 1, x - N + 2, ..., x - 1\}$, either the packet is lost, or if it triggers an ACK that acknowledges sequence number $x - N + 1$, then this ACK is lost.

  Either of these events would freeze the advancement of the right edge of the sender window beyond x, and no new packets with sequence number greater than x would be transmitted. However these events correspond to $E4(x,t'')$ and $E6(x,t'')$ which we assume to have not occurred.
- The sender could potentially send packets with sequence numbers $x+1, x+2, ..., x+N-1$ before it retransmitted x, and these were *all lost* before the measurement point could observe them; this corresponds to event $E3(x,t'')$.
- One of the packets with sequence numbers $x+1, x+2, ..., x+N-1$, that the sender could potentially send before retransmitting x, made it to the measurement point. Let p' be the first of these packets. If p' is reordered in position with p by the channel daemon then the retransmitted packet p would not be considered to be OoS by the measurement point. This is event $E5(x,t'')$.
- The sender retransmitted packet sequence number x *before* it could transmit any packet with sequence number $> x$. Because of Assumption 3 (stated in Section V), this retransmission of x *cannot* be triggered by a timeout - since the sender has always data to send, and must be able to send new packets in the period between the initial transmission of x and the triggering of the retransmission timer. Hence, x much have been retransmitted due to the arrival of duplicate ACKs. Also, the duplicate ACKs must be triggered by packets sent prior to the initial transmission of x, since the sender did not send any new packets before it retransmitted x. If packets sent prior to sequence number x trigger ACKs for x at the receiver, this must be because the sequence numbers of these packets have already been received, and the ACKs for x are an indication that the receiver expects to see a *new* sequence number x. This is a case of *successive fast retransmit*, and the retransmission of x triggered by this event will not be detected as out-of-sequence. This corresponds to event $E7(x,t'')$.

Hence, we have shown that a retransmitted or reordered packet is not detected at the measurement point only if either of the above mentioned events occur for any value of N of the $FastRetx(N)$ protocol.

□

VIII. Implications for TCP

We have verified the classification rules with the end-points implementing simple protocols such as $GB(N)$ and $FastRetx(N)$. We now discuss the implications of our observations if the end-points implement the TCP protocol. We focus on $FastRetx(N)$ since by design it more closely follows TCP behavior, and apply what we have learned from the verification of classification rules for this protocol to two flavors of TCP, namely TCP Reno and NewReno.

We first note that with TCP Reno, when events $E3(x,t'), E4(x,t''), E5(x,t'')E6(x,t'')$ occur, a retransmitted packet will not be detected by the measurement point. Figure 9 illustrates these cases, except for $E7(x,t'')$, for TCP Reno[1]. We also observe that each of these events involves the timeout event at the sender. This implies that the sender behavior described in these cases is independent of the TCP flavor - since all TCP congestion control flavors behave in exactly the same manner subsequent to a timeout event. Thus, the occurrence of the events $E3(x,t'), E4(x,t''), E5(x,t'')E6(x,t'')$ would lead to the non-detection of a retransmitted packet even for the TCP New Reno protocol. If the TCP end-points use the SACK option, then by examining the SACK blocks sent by the receiver, a TCP sender can infer exactly which packets are lost in the channel. A TCP sender, that uses the SACK option, will never misinterpret duplicate ACKs (for a new packet) as indicators of packet loss, and hence can never undergo a successive fast retransmit. Thus, event $E7(x,t'')$ can not occur if the end-points use TCP SACK.

[1]The sequence of events leading to $E7$ are intricate, and we exclude the example due to space limitations, however it is presented in the extended version of this paper.

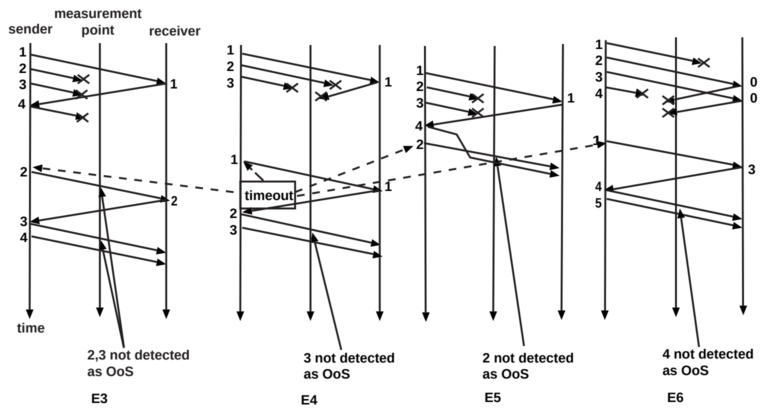

Fig. 9. Undetected retransmissions in TCP Reno

A TCP sender has a dynamically changing window (namely, the additive increase, multiplicative decrease congestion avoidance scheme). This is the primary difference that distinguishes a TCP sender from a $FastRetx(N)$ sender. However, as we have proven in Sections VI and VII about the behavior of $GB(N)$ and $FastRetx(N)$ protocols, the value of the sender window has no impact on the measurement point's ability to detect packet retransmissions and reorderings. Thus we believe, given that $FastRetx(N)$ captures TCP's loss recovery mechanism, that events $E3(x,t'), E4(x,t''), E5(x,t''), E6(x,t'')$, (and $E7(x,t'')$ if the end-points do not use SACK) are the only events for which the classification rules do not detect retransmitted or reordered packets. In future work, we plan to implement the differences between $FastRetx(N)$ and TCP protocols, and carry out a formal verification of TCP to confirm this conjecture.

IX. Conclusions & Future work

In this work, we have proposed a new approach, combining model checking techniques and formal reasoning, to verify the correctness of passive measurements based inference techniques. We analyze the problem of detecting and classifying packet retransmissions and reorderings by a passive monitor in an end-end path. Using the SPIN model checker and by reasoning about network and protocol behavior, we unearth all possible cases in which such packets go undetected at the passive monitor. A natural next step for this work would be to use the insights gained through the formal verification of the inference rules to design new techniques that can capture all packet retransmissions and reorderings. One possible approach would be to drive inferences based on the most likely set of events in the network, using ideas from the body of work on *passive testing* [17], [16] of protocols. While in this work we have considered a specific passive measurements based inference problem, we believe our approach can be applied to other measurement scenarios with similar challenges and constraints.

X. Notes

This work was done when Sharad Jaiswal was a graduate student at UMass Amherst. This research was supported in part by the National Science Foundation under NSF grants ANI-0325868, ANI-0240487, ANI-0085848, and EIA-0080119. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] http://spinroot.com/spin/whatispin.html#d.

[2] Verification and improvement of the sliding window protocol. *Lecture Notes in Computer Science*, 2619:113–127, 2003.

[3] P. Benko and A. Veres. A passive method for estimating end-to-end tcp packet loss. In *Proceedings of IEEE Globecom*, Taipei, Nov. 2002.

[4] D. Bertsekas and R. Gallagher. *Data Networks*. Prentice Hall, 2nd edition, 1991.

[5] K. Bhargavan, D. Obradovic, and C. A. Gunter. Formal verification of standards for distance vector routing protocols. *J. ACM*, 49(4):538–576, 2002.

[6] Y. Dong, X. Du, G. J. Holzmann, and S. A. Smolka. Fighting livelock in the gnu i-protocol: a case study in explicit-state model checking. *STTT*, 4(4):505–528, 2003.

[7] S. Floyd. Tcp and successive fast retransits. Unpublished Note, 1995.

[8] G. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 1st edition, Jan. 2003.

[9] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP connection characteristics through passive measurements (extended version). Technical Report RR03-ATL-070121, Sprint ATL, July 2003.

[10] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Measurement and classification of out-of-sequence packets in a tier-1 ip backbone. In *Proceedings of Infocom*, San Francisco, 2003.

[11] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP connection characteristics through passive measurements. In *Proceedings of Infocom*, Hong Kong, 2004.

[12] S. Jaiswal, G. Iannaccone, J. Kurose, and D. Towsley. Formal verification of passive measurement techniques. Technical Report 05-XX, Computer Science Dept., UMass Amherst, http://gaia.cs.umass.edu, May 2005.

[13] H. Jiang and C. Dovrolis. Passive estimation of TCP round-trip times. *ACM Computer Communication Review*, 32(3), July 2002.

[14] S. Katti, D. Katabi, C. Blake, E. Kohler, and J. Stauss. M&M: A passive toolkit for measuring, tracking and correlating path characteristics. In *Proceedings of ACM Internet Measurements Conference*, Oct. 2004.

[15] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet*. Pearson Benjamin Cummings, 2004.

[16] D. Lee, D. Chen, R. Hao, R. Miller, J. Wu, and X. Yin. A formal approach for passive testing of protocol portions. In *IEEE International Conference on Network Protocols*, Nov. 2002.

[17] D. Lee and M. Yannakkis. Principles and methods of testing finite state machines - a survey. *Proceedings of the IEEE*, 84, Aug. 1996.

[18] R. Miller. Passive testing of networks using a CFSM specification. In *Proceedings of the IEEE International Performance, Computing and Communications Conference*, Feb. 1998.

[19] M. Musuvathi and D. R. Engler. Model checking large network protocol implementations. In *NSDI*, pages 155–168, 2004.

[20] M. Musuvathi, D. Y. W. Park, A. Chou, D. R. Engler, and D. L. Dill. Cmc: A pragmatic approach to model checking real code. In *OSDI*, 2002.

[21] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, June 1999.

[22] M. Smith. Formal verification of communication protocols. In *Formal Description Techniques IX: Theory, Applications, and Tools FORTE/PSTV'96: Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification*, 1996.

[23] M. A. Smith and K. K. Ramakrishnan. Formal specification and verification of safety and performance of tcp selective acknowledgment. *IEEE/ACM Trans. Netw.*, 10(2):193–207, 2002.

[24] A. S. Tanenbaum. *Computer Networks*. Prentice-Hall, 1988.

[25] B. Veal, K. Li, and D. K. Lowenthal. New methods for passive estimation of tcp round-trip times. In *PAM*, pages 121–134, 2005.

[26] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of internet flow rates. In *Proceedings of ACM Sigcomm*, Aug. 2002.