Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

PHRASE BASED BROWSING FOR SIMULATION TRACES OF NETWORK PROTOCOLS

Nathan J. Schmidt
Peter Kemper

Department of Computer Science
College of William and Mary
Williamsburg, VA 23185, U.S.A.

ABSTRACT

Most discrete event simulation frameworks are able to output simulation runs as a trace. The Network Simulator 2 (NS2) is a prominent example that does so to decouple generation of dynamic behavior from its evaluation. If a modeler is interested in the specific details and confronted with lengthy traces from simulation runs, support is needed to identify relevant pieces of information. In this paper, we present a new phrase-based browser that has its roots in information retrieval, language acquisition and text compression which is refined to work with trace data derived from simulation models. The browser is a new navigation feature of Traviando, a trace visualizer and analyzer for simulation traces. The browsing technique allows a modeler to investigate particular patterns seen in a trace, that may be of interest due to their frequent or rare occurrence. We demonstrate how this approach applies to traces generated with NS2.

1 INTRODUCTION

Simulation of stochastic models of discrete event systems is a method that is broadly applied in the performance and dependability evaluation of systems. Examples include network systems, manufacturing and production systems, and computational biology and bioinformatics systems to name a few.

A simulator generates simulation runs that mimic the dynamic behavior of the system under study to provide some sample statistical data for evaluation. Most simulation software has integrated, on-the-fly statistical evaluation to avoid external storage of simulation runs as trace data. The Network Simulator 2 (NS2), however, is a notable exception. NS2 separates the process of trace data generation from statistical evaluation.

NS2 is a large, open source network simulator that is freely available from the University of South Carolina `<www.isi.edu/nsnam/ns>`. NS2 is well known and used extensively in the networking research community. The simulator is frequently used to build traffic models, as in the paper by Lang, Branch, and Armitage (2004), or to test new network protocols, as in (Appenzeller, Keslassy, and McKeown 2004) and (Zapotoczky and Wolter 2008).

In this paper, we will focus on NS2 simulation models and NS2 generated simulation traces in the NS2 Nam format. The contribution of this paper is to show how NS2 traces can be transformed and loaded into Traviando to analyze, visualize and browse the trace quickly, easily, and efficiently. Phrase based browsing is a new feature in Traviando; it reveals details of the dynamic behavior of a model quicker and more easily than is usually possible using highly aggregated performance measures or visual inspection of the NS2 Nam visualizer. The trace browser is based on Nevill-Mannings Sequitur algorithm (Nevill-Manning and Witten 1997), which is able to compute a hierarchical structure based on event labels in linear time. It performs a single scan of the trace and creates the data structures on the fly. The key benefit of the browser is to show events of interest in the specific context they occur in a trace while also grouping equivalent subsequences of events such that a modeler can investigate similar locations in a trace at the same time instead of checking one after another. This approach is implemented in Traviando (Kemper and Tepper 2006b) and readily applies to traces generated by other simulators, such as Möbius (Deavours et al. 2002).

The rest of the paper is structured as follows. In Section 2, we define some terminology with respect to simulation traces. In Section 3, we describe a few characteristics of NS2 and how we obtain and transform NS2 traces into a set of interacting processes which is required by Traviando's XML trace format. Section 4 presents the new phrase-based browser and its underlying technology. In Section 5 we illustrate how phrase-based browsing works for two example models. We evaluate the approach in Section 6 and conclude in Section 7.

2 DEFINITIONS

A trace is a sequence $\sigma = s_0 e_1 s_1 \dots e_n s_n$ of states $s_0, \dots, s_n \in S$ and events $e_1, \dots, e_n \in E$ over some (finite or infinite) sets S, E for an arbitrary but fixed $n \in \mathbf{N}$.

For an event e_i that occurs at position i in σ, let a labeling function $l(e_i) = e$ denote which event $e \in E$ is seen. Furthermore event e_i may carry additional information as a time stamp $t(e_i)$ or other data $d(e_i)$ which is used for instance to make events carry some associated packet header information for traces of simulation models of network protocols. We assume that the time stamps form a monotonously non-decreasing sequence of real-values and that the sequence starts off at time point 0. With this assumption we can define the incremental delay of any event to its predecessors as $\delta t(e_i) = t(e_i) - t(e_{i-1})$ if $i > 1$ and $\delta t(e_i) = t(e_i)$ for $i = 1$. Note that states are irrelevant in the following formal treatment, yet important for a better understanding why the system behaves as observed in σ. Hence, we keep states within our considerations. The length of $\sigma = s_0 e_1 s_1 \dots s_n$ is defined as $|\sigma| = n = \#events$. Let $L = \{l(e_1), \dots, l(e_n)\}$ denote the set of all labels. So $|L| \leq n$ with the expectation that usually $|L| << n$, i.e., events share labels or events of same kind (label) occur repeatedly in σ. For $\sigma = s_0 e_1 s_1 \dots s_n$ and $0 \leq i < j \leq n$, we define a projection or substring operation as $\sigma_{i,j} = s_i e_{i+1} \dots s_j$. We define an equivalence $\approx$ for sequences based on the sequence of labels that we observe for their events. Two sequences σ and σ' of equal length n are equivalent, $\sigma \approx \sigma'$, if $l(e_i) = l(e'_i)$ for $i = 1, \dots, n$. A substring $\sigma_{i,j}$ may have many other substrings in σ that are equivalent. Let $A = \{\sigma_{i_1,j_1}, \dots, \sigma_{i_k,j_k}\}$ be a set of substrings of σ that are all equivalent. The algorithm that we will use to generate such sets will ensure that any pair of elements of a set A does not overlap. We want to provide some timing information with A that we derive from its constituents. We generalize the definition of $\delta t()$ to substrings as follows. For any $\sigma_{i,j}$,

$$\delta t(\sigma_{i,j}) = \begin{cases} t(e_j) - t(e_i) & \text{if } 0 < i < j \\ t(e_j) & \text{if } 0 = i < j \\ 0 & \text{if } i = j \end{cases}$$

Based on this definition, we can define for any such that $A \neq \emptyset$ common statistical measures including

$$\begin{aligned} min(A) &= min\{\delta t(\sigma_{i_l,j_l}) | 1 \leq l \leq k, \sigma_{i_l,j_l} \in A\} \\ max(A) &= max\{\delta t(\sigma_{i_l,j_l}) | 1 \leq l \leq k, \sigma_{i_l,j_l} \in A\} \\ mean(A) &= \sum_{l=1}^{k} \delta t(\sigma_{i_l,j_l}) / k \\ var(A) &= k^{-1} \sum_{l=1}^{k} (\delta t(\sigma_{i_l,j_l})^2) - mean(A)^2 \end{aligned}$$

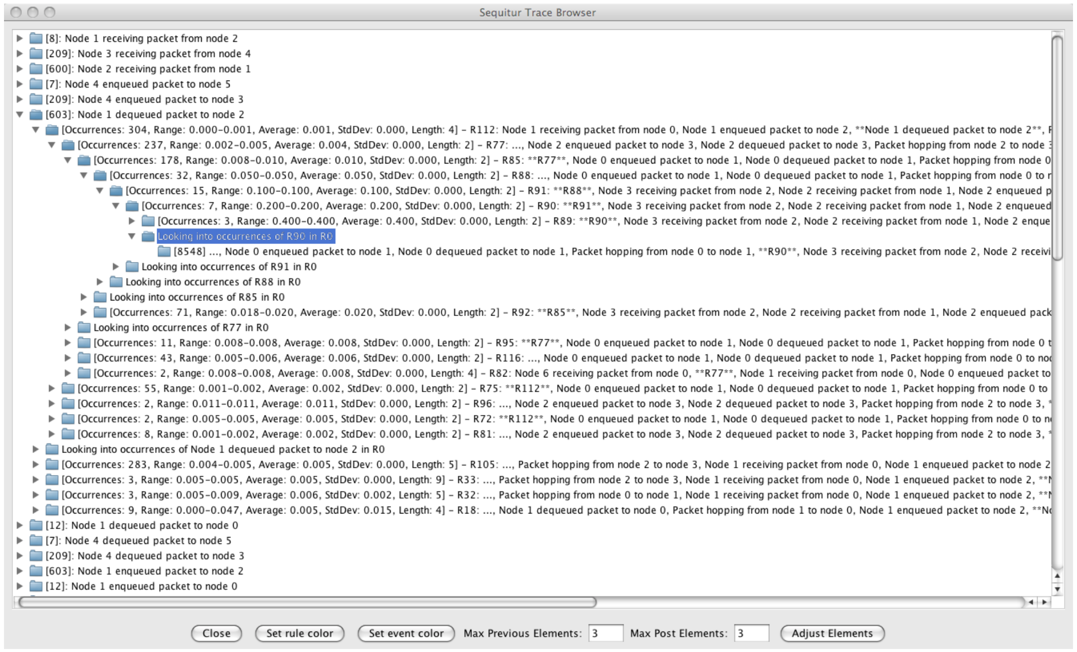
These values give some guidance to a modeler if elements of A are reasonably similar with respect to their timing.

3 NS2 AND NS TRACES

NS2 is a popular simulation tool in the networking research community used to evaluate current protocols and develop new ones. NS2 has a rich set of available components and protocols that can be plugged in as models for network stack layers. Any of these models can be swapped out with a new one that implements a different protocol with relative ease. Using data generated from an NS2 simulation is, unfortunately, not as easy. Any output generated by NS2 during a simulation is directed to the terminal or to a specified trace file. When the simulation has completed, the output, or trace file, can be either examined to view the results and specific details of the simulation or processed by some user given scripts to accumulate data for common performance figures like throughput or the number of packets being dropped over time.

While NS2 does not support any graphical results of a simulation run by itself it does have the Nam visualizer. The Nam visualizer uses a Nam trace file, which is generated during the simulation run, to create a visual representation of the simulation and animates the behavior. Figure 2 shows the Nam visualization for an example considered in Section 5. While the visualizer is helpful to ensure that a simulation is working overall, it does not present the finer details that are produced from the simulation. When checking and debugging a model, these details need to be analyzed in order to guarantee correctness of the model. Simulation results that have been obtained with NS2 and that are published in the literature are usually derived with shell/Python/Pearl scripts that a modeler develops in an adhoc fashion and with possibly a lot of similar work across the NS2 user community. See Evan Jones' website <evanjones.ca/ns2.html> as an example of work in this direction with Jones making his personal efforts available to the community. Those scripts generate input for programs like Gnuplot to generate graphs reported in the literature.

Recent work by Cicconetti, Mingozzi, and Stea (2006) has progressed ones ability to collect data and statistical information from NS2. They contribute a framework that allows NS2 users to collect data and relevant measures with the implementation of their C++ class, *Stat*. Additionally, they provide two more programs that perform a series of runs with a simulation model in order to get the recorded data to a specified level of accuracy. The main deterrence here, however, is that NS2 users must add additional support into their existing simulation models for *Stat*.



```
▶ [8]: Node 1 receiving packet from node 2
▶ [209]: Node 3 receiving packet from node 4
▶ [600]: Node 2 receiving packet from node 1
▶ [7]: Node 4 enqueued packet to node 5
▶ [209]: Node 4 enqueued packet to node 3
▼ [603]: Node 1 dequeued packet to node 2
  ▼ [Occurrences: 304, Range: 0.000-0.001, Average: 0.001, StdDev: 0.000, Length: 4] – R112: Node 1 receiving packet from node 0, Node 1 enqueued packet to node 2, **Node 1 dequeued packet to node 2**, F
    ▼ [Occurrences: 237, Range: 0.002-0.005, Average: 0.004, StdDev: 0.000, Length: 2] – R77: ..., Node 2 enqueued packet to node 3, Node 2 dequeued packet to node 3, Packet hopping from node 2 to node 3
      ▼ [Occurrences: 178, Range: 0.008-0.010, Average: 0.010, StdDev: 0.000, Length: 2] – R85: **R77**, Node 0 enqueued packet to node 1, Node 0 dequeued packet to node 1, Packet hopping from node 0
        ▼ [Occurrences: 32, Range: 0.050-0.050, Average: 0.050, StdDev: 0.000, Length: 2] – R88: ..., Node 0 enqueued packet to node 1, Node 0 dequeued packet to node 1, Packet hopping from node 0 to n
          ▼ [Occurrences: 15, Range: 0.100-0.100, Average: 0.100, StdDev: 0.000, Length: 2] – R91: **R88**, Node 3 receiving packet from node 2, Node 2 receiving packet from node 1, Node 2 enqueued p
            ▼ [Occurrences: 7, Range: 0.200-0.200, Average: 0.200, StdDev: 0.000, Length: 2] – R90: **R91**, Node 3 receiving packet from node 2, Node 2 receiving packet from node 1, Node 2 enqueued
              ▶ [Occurrences: 3, Range: 0.400-0.400, Average: 0.400, StdDev: 0.000, Length: 2] – R89: **R90**, Node 3 receiving packet from node 2, Node 2 receiving packet from node 1, Node 2 enque
              ▼ Looking into occurrences of R90 in R0
                  [8548] ..., Node 0 enqueued packet to node 1, Node 0 dequeued packet to node 1, Packet hopping from node 0 to node 1, **R90**, Node 3 receiving packet from node 2, Node 2 receivi
            ▶ Looking into occurrences of R91 in R0
          ▶ Looking into occurrences of R88 in R0
        ▶ Looking into occurrences of R85 in R0
        ▶ [Occurrences: 71, Range: 0.018-0.020, Average: 0.020, StdDev: 0.000, Length: 2] – R92: **R85**, Node 3 receiving packet from node 2, Node 2 receiving packet from node 1, Node 2 enqueued pack
      ▶ Looking into occurrences of R77 in R0
      ▶ [Occurrences: 11, Range: 0.008-0.008, Average: 0.008, StdDev: 0.000, Length: 2] – R95: **R77**, Node 0 enqueued packet to node 1, Node 0 dequeued packet to node 1, Packet hopping from node 0 t
      ▶ [Occurrences: 43, Range: 0.005-0.006, Average: 0.006, StdDev: 0.000, Length: 2] – R116: ..., Node 0 enqueued packet to node 1, Node 0 dequeued packet to node 1, Packet hopping from node 0 to noc
      ▶ [Occurrences: 2, Range: 0.008-0.008, Average: 0.008, StdDev: 0.000, Length: 4] – R82: Node 6 receiving packet from node 0, **R77**, Node 1 receiving packet from node 0, Node 0 enqueued packet to
    ▶ [Occurrences: 55, Range: 0.001-0.002, Average: 0.002, StdDev: 0.000, Length: 2] – R75: **R112**, Node 0 enqueued packet to node 1, Node 0 dequeued packet to node 1, Packet hopping from node 0 to
    ▶ [Occurrences: 2, Range: 0.011-0.011, Average: 0.011, StdDev: 0.000, Length: 2] – R96: ..., Node 2 enqueued packet to node 3, Node 2 dequeued packet to node 3, Packet hopping from node 2 to node 3, *
    ▶ [Occurrences: 2, Range: 0.005-0.005, Average: 0.005, StdDev: 0.000, Length: 2] – R72: **R112**, Node 0 enqueued packet to node 1, Node 0 dequeued packet to node 1, Packet hopping from node 0 to n
    ▶ [Occurrences: 8, Range: 0.001-0.002, Average: 0.002, StdDev: 0.000, Length: 2] – R81: ..., Node 2 enqueued packet to node 3, Node 2 dequeued packet to node 3, Packet hopping from node 2 to node 3, *
  ▶ Looking into occurrences of Node 1 dequeued packet to node 2 in R0
  ▶ [Occurrences: 283, Range: 0.004-0.005, Average: 0.005, StdDev: 0.000, Length: 5] – R105: ..., Packet hopping from node 2 to node 3, Node 1 receiving packet from node 0, Node 1 enqueued packet to node 2
  ▶ [Occurrences: 3, Range: 0.005-0.005, Average: 0.005, StdDev: 0.000, Length: 9] – R33: ..., Packet hopping from node 2 to node 3, Node 1 receiving packet from node 0, Node 1 enqueued packet to node 2, **N
  ▶ [Occurrences: 3, Range: 0.005-0.009, Average: 0.006, StdDev: 0.002, Length: 5] – R32: ..., Packet hopping from node 0 to node 1, Node 1 receiving packet from node 0, Node 1 enqueued packet to node 2, **N
  ▶ [Occurrences: 9, Range: 0.000-0.047, Average: 0.005, StdDev: 0.015, Length: 4] – R18: ..., Node 1 dequeued packet to node 0, Packet hopping from node 1 to node 0, Node 1 enqueued packet to node 2, **No
▶ [12]: Node 1 dequeued packet to node 0
▶ [7]: Node 4 dequeued packet to node 5
▶ [209]: Node 4 dequeued packet to node 3
▶ [603]: Node 1 enqueued packet to node 2
▶ [12]: Node 1 enqueued packet to node 0

Max Previous Elements: 3   Max Post Elements: 3
```

Figure 1: Example of an expanded tree in Traviando's Trace Browser

3.1 NS2 Trace Formats

NS2 currently supports a number of different types of trace files. In addition to its own format, NS2 also has the Nam trace format, which contains the necessary information from the simulation to drive the Nam visualizer. Both of these trace formats are very specific when it comes to giving details about the events that occur during an NS2 simulation. State information generated by both trace formats generally consists of updates on whether a node, or link, is up or down. Both of these formats are open formats and are published in the NS2 manual (Fall and Varadhan 2008). However, there are also further formats such as the cmu-trace and traffictrace formats. The trace formats mentioned are included in *most* protocols and new protocols are adapting the trace formats as they are added. All existing trace formats can be enabled or disabled in the OTcl script that describes a model. Figure 3 shows some lines extracted from Nam formatted traces that we generated with NS2 for the examples in Section 5. The format is ascii based and line oriented to simplify processing with scripting languages. Single letters serve as keywords to specify the meaning of subsequent numerical values.

We chose to use the Nam trace format for three reasons. A Nam trace contains sufficient information that is needed to track a packet's path as it moves from node to node. It is commonly supported in modeling with NS2 due to the fact that the Nam visualizer requires it and that visualizer is the only immediate way to receive visual feedback from a simulation. Finally, the Nam trace format is open and its documentation is available within the NS2 manual.

3.2 Reformatting Nam Traces

In order to view the NS2 simulation in Traviando, we need to format the NS2 Nam trace data into Traviando's open XML format as documented in (Kemper and Tepper 2006a). The XML format requires the trace data to be organized into processes, actions that are either performed by a single process or by several processes in a jointly manner, and state variables, whose values change as an effect of the actions performed in a simulation run. The sets of processes, actions and state variables are finite and declared ahead of the sequence of events that document the events performed in a simulation. An event denotes which action (label) is performed and can carry additional information as new values for state variables, a time stamp and additional optional information.

In order to match this with Nam traces, we need to map NS2 nodes, links and events to processes, actions and state variables. We derive two different sets of processes from a Nam trace. For the first set, we define one process for each node in the NS2 network. This allows one to view all actions that occur at a node in a single, vertical timeline in the MSC visualization of Traviando. For the second set of processes we define one process for each link between nodes. Links need to be represented as processes because packets need to belong to a process when they are transitioning between two nodes. Links also carry state information and can be either up or down. The link (process) describes the shared communication medium. For wireless models, we introduce an additional process that models the single shared communication medium and nevertheless keep the link processes since those are present in the trace data as well. Link processes can have the following actions: send, receive, up and down. The actions up and down happen if a link between two nodes is restored or severed and the other two actions represent a link receiving a packet from a node or sending a packet to a node. Send and receive actions are jointly performed by a link and a corresponding node process. So, node processes have send and receive actions and also up and down actions that correspond to when a node comes online or goes offline. In addition to that a node has further actions: enqueue, dequeue, and drop. Each node on the network has a queue. This queue is of a certain size and acts as a buffer for packets waiting to be sent. So, for example, when a node receives a packet that needs to be forwarded along to another node, it receives the packet and then enqueues the packet onto its sending queue. The packet waits in the queue until it is dequeued at which time it is then sent to link that connects the two nodes. The drop action occurs when the queue at a node is full and additional packets are received and need to be sent to other nodes. The type of queue each node uses determines which packet is dropped if too many packets are enqueued at once. Time stamps associated with events directly carry over from the Nam format to Traviando's format while packet ids are mapped to additional optional information associated with events in Traviando's format.

Lastly variables are defined for each process. Each link process only has a single variable that maintains its state, which is set to one if operational or zero if it is down. Node processes also have a similar state variable keeping track of whether or not the node is online or offline. In addition, node processes also have the variables: *sendcounter*, *drop-counter*, *and receivecounter*. These variables keep track of the number of sent, dropped, and received packets for a node.

All of this is done in *Nam2Traviando*. *Nam2Traviando* is currently a stand alone Java program that converts Nam trace files into the Traviando's open XML format with no additional work. After the XML file is produced it can be opened from within Traviando to view the NS2 trace.

4 PHRASE BASED BROWSING IN TRAVIANDO

A simulation usually generates large amounts of trace data which creates a demand for some form of automated support to analyze this data. NS2 is a prominent example for this effect. Using the NS2 Nam trace format, a fairly simple seven node network simulation generates almost 11000 lines of trace data in under five simulated seconds. If analysis shall go beyond a calculation of highly aggregated performance figures like buffer occupancy or throughput, the usual scripting efforts made by NS2 users become a real challenge. We propose a phrase-based browsing technique to help a user investigate the detailed behavior of a simulation run and, in particular, to help debug a simulation model. For instance, if a modeler is interested in what events happen before a packet is dropped at some node, then he/she can visit sequentially all locations in a trace σ where the corresponding event of a packet loss takes place. If those locations are numerous, it is very likely that neighboring events are of similar kind. Since for most traces, $|L| << n$, i.e., a simulation model performs events with labels selected from a relatively small reservoir L, a trace σ contains many substrings $\sigma_{i,j}$ that are equivalent with respect to their labels. This has potential for supporting a browser that shows a particular event as it appears at different locations in σ and groups locations if they belong to equivalent substrings. The key concept is to help a modeler see events in the context of other events in the trace and to group those locations according to contexts (or substrings) that are equivalent. Let l be the label of interest and let l be present in σ', a substring of σ, and let σ' be present k times at different locations of σ, i.e., there is $A = \{\sigma_{i,j} | \sigma_{i,j} \approx \sigma'\}$ of cardinality k. The benefit in this case would be that it is possible to inspect all k locations at once based on σ'. If the sequence of labels in σ' gives sufficient information to help the modeler decide if these locations require further attention or can safely be ignored, then this saves an inspection of $k-1$ locations of same kind. Note that the notion of equivalence is based on labels only. To provide more information associated with a set A of substrings, we added its cardinality, the length of any of its elements, and timing information $min(A)$, $max(A)$, $mean(A)$, and $var(A)$ to the representation of A in the browser. This additional information helps a modeler recognize if the timing characteristics of substrings in A are rather homogenous or differ greatly. If A requires further attention, the browser allows a modeler to partition A into smaller subsets, investigate those subsets and to iterate on this till a particular, single location in σ is reached, where full detailed information on states, time stamps, and context is available to the full level of detail contained in σ. This approach has the potential to largely improve productivity in tracking errors in simulation runs, be it obvious ones whose effects are seen frequently in a trace or the rare ones, 'the needles in the haystack', that are difficult to find.

4.1 Sequitur Browser Details

The Sequitur Browser uses a tree type structure to browse the trace file for each individual event. Hence the first level lists all of the different events possible in the trace with the corresponding number of occurrences in the trace. These items serve as roots for individual trees whose nodes expand on the context the event associated with the root nodes appears in. An example of this can be seen in Figure 1. When browsing an NS2 trace, these events consist of the nodes and links on the network sending, receiving, enqueueing, dequeueing, and dropping packets.

Subsequent nested levels at each tree display groups of locations in the trace where the event at level 1 is present. The tree-type structure is based on a grammar with a set of rules that characterize σ as the only word of its language. Following a path in a tree and expanding its nodes towards a leaf node corresponds to navigating in the rule set of that grammar. Nodes at some level $k > 1$ are accomplished by using the phrases, or rules, which group together equivalent event sequences $\sigma_{i,j}$ in the trace file. Numeric information that is given with any particular node at a level $k > 1$ represents the number of locations in σ that rule represents, the length of the substring as well as additional timing information. In its current stage, the browser also shows which rules of the grammar are used. Rules are shown in the format R#, where # represents the number of the rule. Following the rule name is a selection of part of the contents from the specific rule. These contents include simulation events from the rule and at least one occurrence of the event from level 1. The event from level 1 is surrounded by double stars (**) to draw the modeler's attention to it because this is the event that the modeler is searching for. The selection, by default, displays the previous three events and the next three events that occur before and after an occurrence of the event from level 1. Depending on screen size and how much surrounding information is desired, any number of previous events and events after the level 1 event can be displayed.

Navigating from a node at level k to a node at level $k+1$ refines the set of locations seen at level k into subsets of smaller cardinality but longer substrings (more context) that is taken into account. This process can continue until a leaf node is reached. It is important to note that the level prior to a leaf node in the tree alerts the user to the face that the following leaf nodes are occurrences from R0, the start rule. The leaf nodes below this show all occurrences of the rule/event represented one level above this node. A leaf node represents a single location in the trace that corresponds to a substring of the rule above the alert (which is the one two levels above the leaf node). Lastly, the number at the beginning of the leaf node represents the location within the trace where the rule/event from two levels above begins. An expanded tree showing intermediate levels and leaf nodes can be seen in Figure 1.

The textual display in the browser is accompanied by a message sequence chart (MSC) like visualization in Traviando that is more instructive and shown on demand with a right click on the corresponding tree node in the browser. The MSC visualization shows processes with corresponding time lines (vertical lines) and events either as events local at some process (points) or events performed jointly by several processes (horizontal lines, directed or undirected). In the MSC visualization, events of the rule are highlighted in dark red, the particular event of interest in bright red. If a node represents more than one location in the trace, the first matching location is taken as a representative of that set.

4.2 Underlying Technology

In order to make this approach operational, some technical issues need to be resolved. Obviously the length of any considered σ' relates to the cardinality of the corresponding set of locations A. In order to have a large cardinality of A, σ' needs to be short, yet to be able to recognize interesting behavior in σ' its length cannot be too short. It remains to be determined which substrings shall be considered for browsing and if those substrings and their groups shall be computed on demand or once and independently of what events are of interest. While the former has more potential benefits from a users perspective, the latter will turn out to be efficiently possible in practice based on the Sequitur algorithm from Nevill-Manning (Nevill-Manning and Witten 1997). We use the Sequitur algorithm to generate a grammar that serves as a rule base for navigation of the entire trace. The rules of the grammar are what group similar sequences of events together into rules. The Sequitur algorithm is known for generating a concise grammar from a sequence of symbols with the in a single scan with a time complexity that is linear in the length of the sequence and modest memory requirements. For more information on the Sequitur algorithm please see Nevill-Manning's PhD thesis (Nevill-Manning 1996), available with more information at `<sequitur.info>`.

4.3 Other Features of Traviando

In addition to the new phrase based browser, Traviando provides a variety of features to consider the detailed behavior of a simulation model. For a single trace, it provides basic statistical information on the number of changes made to state variables and on the number of times events occur. It sheds light on the progressive or repetitive character of a trace, i.e., if states are frequently revisited and the behavior is cyclic or if a simulation never returns to a previously reached state. There are two ways to illustrate this, one is with the help of a measure called progress (Kemper 2007), which shows distinctive patterns that allow a simple classification of models. Another visualization plots the state indices along with the trace. Both features help to identify certain types of errors in simulation models. All features of this category aim at general characteristics of traces.

In order to check particular properties, Traviando supports LTL model checking of traces. Formulas are specified

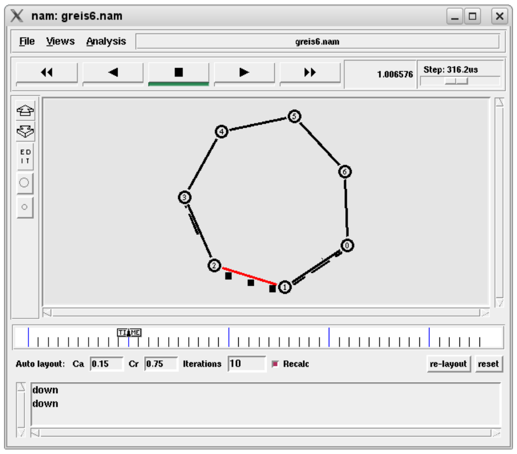
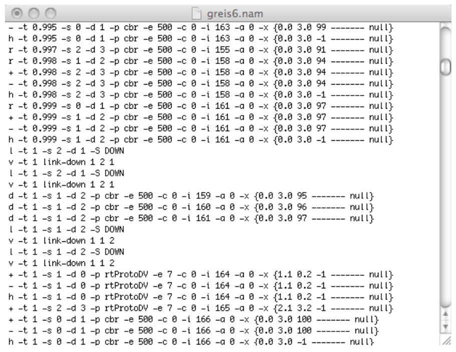
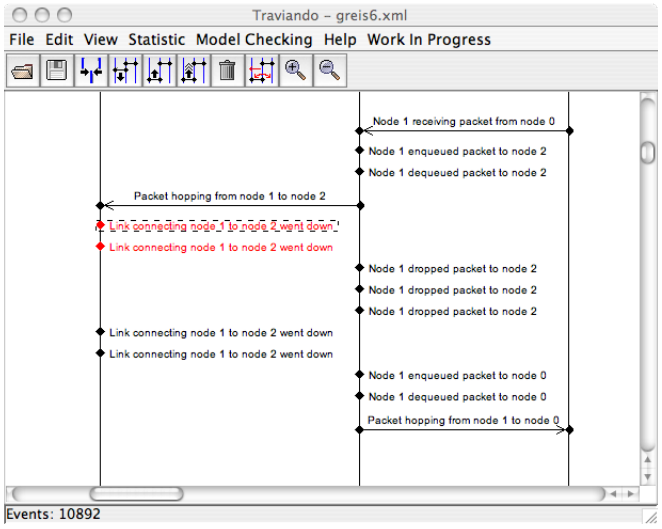


Figure 2: Example of the Nam visualizer.



```
- -t 0.995 -s 0 -d 1 -p cbr -e 500 -c 0 -i 163 -a 0 -x {0.0 3.0 99 ------- null}
h -t 0.995 -s 0 -d 1 -p cbr -e 500 -c 0 -i 163 -a 0 -x {0.0 3.0 -1 ------- null}
r -t 0.997 -s 2 -d 3 -p cbr -e 500 -c 0 -i 155 -a 0 -x {0.0 3.0 91 ------- null}
r -t 0.998 -s 1 -d 2 -p cbr -e 500 -c 0 -i 158 -a 0 -x {0.0 3.0 94 ------- null}
+ -t 0.998 -s 2 -d 3 -p cbr -e 500 -c 0 -i 158 -a 0 -x {0.0 3.0 94 ------- null}
- -t 0.998 -s 2 -d 3 -p cbr -e 500 -c 0 -i 158 -a 0 -x {0.0 3.0 94 ------- null}
h -t 0.998 -s 2 -d 3 -p cbr -e 500 -c 0 -i 158 -a 0 -x {0.0 3.0 -1 ------- null}
r -t 0.999 -s 0 -d 1 -p cbr -e 500 -c 0 -i 161 -a 0 -x {0.0 3.0 97 ------- null}
+ -t 0.999 -s 1 -d 2 -p cbr -e 500 -c 0 -i 161 -a 0 -x {0.0 3.0 97 ------- null}
- -t 0.999 -s 1 -d 2 -p cbr -e 500 -c 0 -i 161 -a 0 -x {0.0 3.0 97 ------- null}
h -t 0.999 -s 1 -d 2 -p cbr -e 500 -c 0 -i 161 -a 0 -x {0.0 3.0 -1 ------- null}
l -t 1 -s 2 -d 1 -S DOWN
v -t 1 link-down 1 2 1
l -t 1 -s 2 -d 1 -S DOWN
v -t 1 link-down 1 2 1
d -t 1 -s 1 -d 2 -p cbr -e 500 -c 0 -i 159 -a 0 -x {0.0 3.0 95 ------- null}
d -t 1 -s 1 -d 2 -p cbr -e 500 -c 0 -i 160 -a 0 -x {0.0 3.0 96 ------- null}
d -t 1 -s 1 -d 2 -p cbr -e 500 -c 0 -i 161 -a 0 -x {0.0 3.0 97 ------- null}
l -t 1 -s 1 -d 2 -S DOWN
v -t 1 link-down 1 1 2
l -t 1 -s 1 -d 2 -S DOWN
v -t 1 link-down 1 1 2
+ -t 1 -s 1 -d 0 -p rtProtoDV -e 7 -c 0 -i 164 -a 0 -x {1.1 0.2 -1 ------- null}
- -t 1 -s 1 -d 0 -p rtProtoDV -e 7 -c 0 -i 164 -a 0 -x {1.1 0.2 -1 ------- null}
h -t 1 -s 1 -d 0 -p rtProtoDV -e 7 -c 0 -i 164 -a 0 -x {1.1 0.2 -1 ------- null}
+ -t 1 -s 2 -d 3 -p rtProtoDV -e 7 -c 0 -i 165 -a 0 -x {2.1 3.2 -1 ------- null}
+ -t 1 -s 0 -d 1 -p cbr -e 500 -c 0 -i 166 -a 0 -x {0.0 3.0 100 ------- null}
- -t 1 -s 0 -d 1 -p cbr -e 500 -c 0 -i 166 -a 0 -x {0.0 3.0 100 ------- null}
h -t 1 -s 0 -d 1 -p cbr -e 500 -c 0 -i 166 -a 0 -x {0.0 3.0 -1 ------- null}
```

Figure 3: Lines in Nam trace file of simple network model.


Traviando – greis6.xml
File Edit View Statistic Model Checking Help Work In Progress

```
Node 1 receiving packet from node 0
Node 1 enqueued packet to node 2
Node 1 dequeued packet to node 2
Packet hopping from node 1 to node 2
Link connecting node 1 to node 2 went down
Link connecting node 1 to node 2 went down
Node 1 dropped packet to node 2
Node 1 dropped packet to node 2
Node 1 dropped packet to node 2
Link connecting node 1 to node 2 went down
Link connecting node 1 to node 2 went down
Node 1 enqueued packet to node 0
Node 1 dequeued packet to node 0
Packet hopping from node 1 to node 0
Events: 10892
```

Figure 4: MSC Visualization triggered from Sequitur Trace Browser for the event of a link going down.

in an editor that provides multiple predefined skeletons for commonly applied types of formulas (model checking patterns (Kemper and Tepper 2006b)) which only need to be adjusted by adding appropriate atomic propositions. Model checking results are shown by color highlighting those events in a MSC visualization that lead to states that fulfill a given formula. Colors can be freely associated with formulas and subformulas. The MSC visualization is enhanced with more features to support process interaction models and to highlight congestion of entities at resources (Kemper and Tepper 2005) .

Finally, clustering of a set of simulation traces is supported with respect to a set of given features. The motivation is to help a modeler group traces that result from a series of simulation experiments such that one may examine a group of traces with a particular property or on traces taken from different groups to investigate the differences.

5 ANALYSIS OF NETWORK PROTOCOL MODELS

5.1 A Simple Network Model

We consider a simple network model taken from Marc Greis' online tutorials that teaches one how to create a larger network topology and how to simulate a link failure with NS2, see tutorial 6 at <www.isi.edu/nsnam/ns>. The Nam visualizer is able to visually display the events of this NS2 simulation graphically to the user. An example of the Nam visualizer running on corresponding trace data can be seen in Figure 2.

In this example, Node 0 is trying to send UDP packets to node 3. At time 1.0 seconds, the link between node 1 and node 2 goes down, which can be seen in Figure 2. Node 0 is forced to reroute it's traffic through nodes 6, 5, and 4 in order to send its packets to node 3. The Nam visualizer is helpful here to show that the experiment is working, but it does not present the finer details that are produced from this simulation. When checking and debugging a model, these details need to be analyzed in order to guarantee correctness. Unfortunately, such details are either not seen or unclear in the Nam visualizer.

The particular NS2 trace of this model that we consider contains 10917 lines. By loading the trace into Traviando we are able to quickly produce graphs calculating statistical information about the number of events and resulting variables in the NS2 trace. We see that of the 823 packets sent by node 0, 812 of them were received by node 3. Node 0 dropped 3 packets and the rest must have been lost in transit. We are using the new Sequitur Trace Browser in Traviando to quickly browse the trace file. The grammar generated by the Sequitur algorithm for this trace contains 120 rules. After expanding a couple of layers in the browser window as shown in Figure 1, we immediately see various repetitious patterns. For example R112, R105, R32, and R18 all contain sequences of events where a node receives a packet, enqueues the packet, dequeues the packet, and then sends the packet to another node. These patterns may have not been detected by the usual means of extracting data from NS2 source files.

We can also immediately find the trace events where packets were dropped when the link connecting nodes 1 and 2 goes down. We do this by looking at level 1 nodes in the Sequitur Trace Browser tree and finding the action that describes that a link goes down. We can then expand this event to see the alert representing rule R0. After selecting the first leaf node below the alert, we immediately jump to that position in the MSC of the trace and the action is highlighted in bright red as shown in Figure 4. Analogously, we can identify events where packets are dropped. Figure 4 shows a location where packets are dropped as well. It is important to point out that the event of a packet being dropped is very rare because only 3 packets are dropped when the link goes down. As a result, the event of a packet being dropped does not belong to a phrase/rule, which is unusual and should draw the modelers attention when the modeler is browsing through the trace file with the Sequitur Trace Browser.

Another interesting observation is that when a link goes down the NS2 Nam trace format dumps four events for this one action as shown in Figure 4. The four events come as a pair of two events that are subsequently repeated once. The pair of events contains of one event to describe that a link from node A to node B goes down and a second event for the link from B to A going down. So links are treated as directed in this model. However it is unclear, why this is reported twice for each direction and with all four events having the same time stamp of 1.0. In the trace file these two events are one after the other and occur two times during the simulation, so they are grouped into R38. R38 occurs twice in the start rule, R0, which we thought to be interesting behavior for NS2 since the second occurrence of R38 is unexpected. We confirmed that this identical event exists by finding it in the Nam trace file generated by NS2 seen in Figure 3. A similar thing happens when the link comes back online. NS2 generates four link ON trace messages when the link comes back online at 2.0 seconds. This information is also visible in the Sequitur Trace Browser. This is interesting because a person using bash scripting to generate data from a trace file may calculate an incorrect count for the number of times a given link goes down or up.

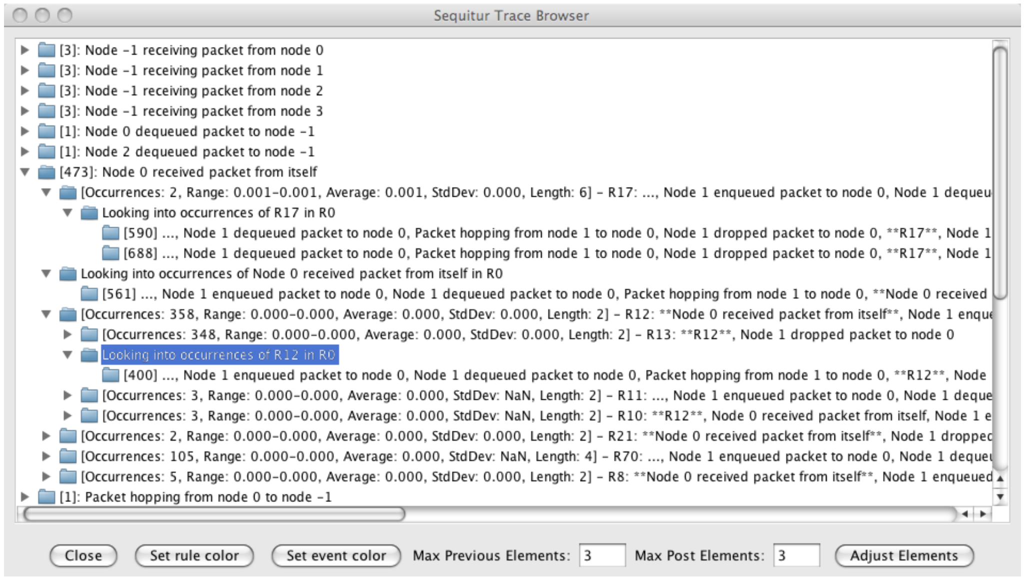
5.2 AN 802.11e Protocol Model

Zapotoczky and Wolter (Zapotoczky and Wolter 2008) increase the performance of the 802.11e protocol by introducing access category shifting. They point out that 802.11e does not give good service quality to low priority class clients when the only clients on the network are the low priority class clients. They propose that for this specific case all the low priority clients should temporarily shift up to a higher priority in order to better utilize the network and achieve a higher throughput. They experiment with this proposed change both with hardware and with simulation and show that they are able to achieve up to 20% higher overall throughput by using access category shifting. For their simulations they use NS2 version 2.31 with the 802.11e EDCA and CFB Simulation Model (Wietholter, Emmelmann, Hoene, and Wolisz 2006). Here we explore one of the trace files from a simulation of this model.

The simulation model represents a wireless network with 4 mobile nodes. Node 1 generates traffic using a CBR generator while node 0 acts as a sink and listens to the traffic. The Nam trace file generated by the simulation is 1960108 lines (130M in size) long, which represents 16.01 seconds worth of data. Here we only examine the first 15000 lines of the trace (about 2.10 seconds) and our observations relate to only those lines. Unfortunately Nam visualizer version 1.13 segmentation faults when attempting to visualize both the shortened trace and the full trace.

After loading the shortened trace into Traviando, we found the following data. Within the first 2.10 seconds, node 1 sends out 4293 packets, nodes 0, 2, and 3 send out 1 packet each, and node 0 receives 473 packets. Wireless networks result in slightly different and particular NS2 models. NS2 references a node with id -1 in the Nam trace. We interpret this node to be the wireless network. For this model, the set of processes in the MSC visualization contains links between each node and node -1 along with a link from node 1 to node 0, so we used a structuring feature of Traviando to combine processes into groups. This is helpful to retain a clear graphical representation in cases like this as it makes logical sense to combine all links and node -1 into one process called Radio. This process then represents the medium when packets are sent wirelessly across the network. One can also see by looking at the event count from the Sequitur Trace Browser that nodes 2 and 3 do not really do much in the simulation. We therefore group them together into one process called Other-Nodes. The resulting MSC is a condensed version of the previous MSC, however, no events have been cut out and the MSC is still complete. The resulting MSC can be seen in Figure 6.

One interesting thing about this simulation is that packets appear to be generated in the medium. The usual order of events consist of: node A sends packet to link AB, link



```
[3]: Node -1 receiving packet from node 0
[3]: Node -1 receiving packet from node 1
[3]: Node -1 receiving packet from node 2
[3]: Node -1 receiving packet from node 3
[1]: Node 0 dequeued packet to node -1
[1]: Node 2 dequeued packet to node -1
[473]: Node 0 received packet from itself
  [Occurrences: 2, Range: 0.001-0.001, Average: 0.001, StdDev: 0.000, Length: 6] - R17: ..., Node 1 enqueued packet to node 0, Node 1 dequeu
    Looking into occurrences of R17 in R0
      [590] ..., Node 1 dequeued packet to node 0, Packet hopping from node 1 to node 0, Node 1 dropped packet to node 0, **R17**, Node 1
      [688] ..., Node 1 dequeued packet to node 0, Packet hopping from node 1 to node 0, Node 1 dropped packet to node 0, **R17**, Node 1
  Looking into occurrences of Node 0 received packet from itself in R0
    [561] ..., Node 1 enqueued packet to node 0, Node 1 dequeued packet to node 0, Packet hopping from node 1 to node 0, **Node 0 received
  [Occurrences: 358, Range: 0.000-0.000, Average: 0.000, StdDev: 0.000, Length: 2] - R12: **Node 0 received packet from itself**, Node 1 enque
    [Occurrences: 348, Range: 0.000-0.000, Average: 0.000, StdDev: 0.000, Length: 2] - R13: **R12**, Node 1 dropped packet to node 0
    Looking into occurrences of R12 in R0
      [400] ..., Node 1 enqueued packet to node 0, Node 1 dequeued packet to node 0, Packet hopping from node 1 to node 0, **R12**, Node
    [Occurrences: 3, Range: 0.000-0.000, Average: 0.000, StdDev: NaN, Length: 2] - R11: ..., Node 1 enqueued packet to node 0, Node 1 deque
    [Occurrences: 3, Range: 0.000-0.000, Average: 0.000, StdDev: NaN, Length: 2] - R10: **R12**, Node 0 received packet from itself, Node 1 e
  [Occurrences: 2, Range: 0.000-0.000, Average: 0.000, StdDev: 0.000, Length: 2] - R21: **Node 0 received packet from itself**, Node 1 droppec
  [Occurrences: 105, Range: 0.000-0.000, Average: 0.000, StdDev: NaN, Length: 4] - R70: ..., Node 1 enqueued packet to node 0, Node 1 dequeu
  [Occurrences: 5, Range: 0.000-0.000, Average: 0.000, StdDev: 0.000, Length: 2] - R8: **Node 0 received packet from itself**, Node 1 enqueued
[1]: Packet hopping from node 0 to node -1
```

Max Previous Elements: 3 Max Post Elements: 3

Figure 5: Sequitur Trace Browser from 802.11e protocol simulation.

AB receives packet and sends it to node B, and finally node B receives the packet. At the beginning of the simulation, this is not the case as node -1 receives packets from nodes 1, 3, and 2 without nodes 1, 3, or 2 sending any packets (this can also be seen in Figure 6). Upon seeing this, we manually examined the Nam trace file to confirm that this was actually happening. For the given model, it is not clear why NS2 performs these events.

Another interesting behavior seen in this simulation is how node 0 receives packets. By looking at both the MSC and at the Sequitur Trace Browser in Figure 5 it appears that all the packets received by node 0 are received from itself. Again, we manually checked the Nam trace file to ensure that this is actually the case. Again, it is not clear why NS2 performs these events.

In summary, trace based analysis and the Sequitur Trace Browser helped us to identify interesting details of simulation traces generated from NS2 models. We did not track down the particular reasons for that behavior within NS2 for the time being, since the main purpose for the discussion of these examples is to illustrate how our approach can be applied in practice.

6 EVALUATION

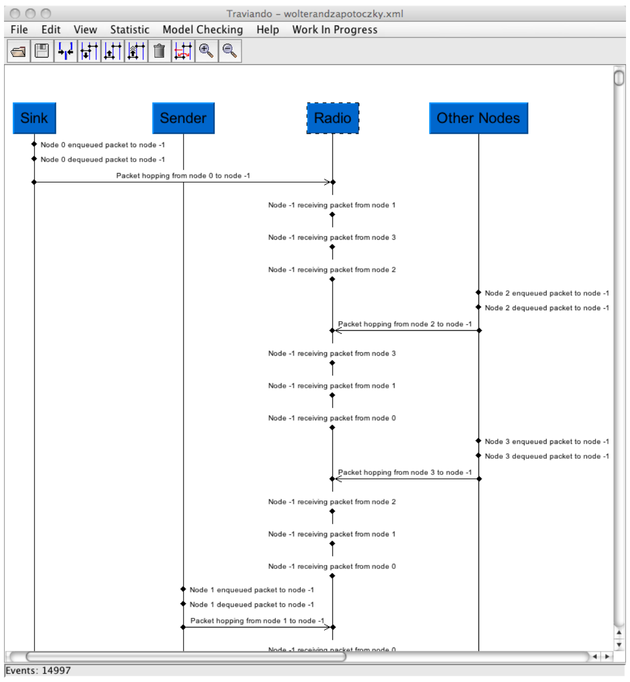
The Sequitur Trace Browser helps to display patterns and repetitions to modelers that might otherwise be missed by the usual scripting methods of collecting and aggregating data. The browser's ability to aggregate data into phrases in linear time and form it into an effective visual representation of what really occurs during the simulation is the browser's primary benefit.

In its current form the Sequitur Trace Browser does not provide the modeler with any control over how the phrases are constructed. As a result, if the modeler is looking for specific repetitious behavior, the browser may or may not group that specific behavior into a phrase. This is the tradeoff that is paid in order to achieve linear complexity and it is important to achieve this when dealing with large trace files, which is usually the case. Sequitur tends to create rather short rules of at least 2 symbols in order to achieve a small rule base which implies that the expansion of rules in the browser tends to duplicate the length of the considered sequence in each step.

At this point, the Sequitur Browser applies to a single trace. The generalization to browse a set of traces is immediate at a conceptual level. However, additional information needs to be incorporated into the rule base to keep track of which trace(s) a substring refers to. We leave this for future work.

7 CONCLUSION

We describe a transformation of NS2 Nam traces of both network and wireless network models into an XML input format for Traviando, a trace visualizer and analyzer. We propose a new phrase-based trace browser that has been integrated into Traviando, which lets a modeler browse and investigate sets of equivalent parts of a simulation trace simultaneously. In combination with other features of Traviando, this browser makes debugging NS2 models



```
Node 0 enqueued packet to node -1
Node 0 dequeued packet to node -1
Packet hopping from node 0 to node -1
Node -1 receiving packet from node 1
Node -1 receiving packet from node 3
Node -1 receiving packet from node 2
Node 2 enqueued packet to node -1
Node 2 dequeued packet to node -1
Packet hopping from node 2 to node -1
Node -1 receiving packet from node 3
Node -1 receiving packet from node 1
Node -1 receiving packet from node 0
Node 3 enqueued packet to node -1
Node 3 dequeued packet to node -1
Packet hopping from node 3 to node -1
Node -1 receiving packet from node 2
Node -1 receiving packet from node 1
Node -1 receiving packet from node 0
Node 1 enqueued packet to node -1
Node 1 dequeued packet to node -1
Packet hopping from node 1 to node -1
Node -1 receiving packet from node 0
Events: 14997
```

Figure 6: Condensed MSC.

easier and more productive. We report on our initial findings when applying our technique to traces obtained from NS2 models found in the literature, one is that of a Token Ring type network protocol and a second one that models aspects of the 802.11e protocol for wireless networks. In both cases we were able to quickly see behavior from the simulation that motivates further investigations and is not trivially explained.

REFERENCES

Appenzeller, G., I. Keslassy, and N. McKeown. 2004. Sizing router buffers. *SIGCOMM Comput. Commun. Rev.* 34 (4): 281–292.

Cicconetti, C., E. Mingozzi, and G. Stea. 2006. An integrated framework for enabling effective data collection and statistical analysis with ns-2. In *WNS2 '06: Proceeding from the 2006 workshop on ns-2: the IP network simulator*, 11. New York, NY, USA: ACM.

Deavours, D. D., G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster. 2002. The Möbius framework and its implementation. *IEEE Trans. Software Eng.* 28 (10): 956–969.

Fall, K., and K. Varadhan. 2008. The ns manual (formerly ns notes and documention). Technical report, UC Berkeley and LBL and USC/ISI and Xerox PARC.

Kemper, P. 2007. A trace-based visual inspection technique to detect errors in simulation models. In *Winter Simulation Conference*, 747–755: ACM.

Kemper, P., and C. Tepper. 2005. Trace based analysis of process interaction models. In *Winter Simulation Conference*, 427–436: ACM.

Kemper, P., and C. Tepper. 2006a. Trace analysis - gain insight through modelchecking and cycle reduction. Technical Report 06007, Universitat Dortmund, SFB 559, Modeling Large Networks in Logistics.

Kemper, P., and C. Tepper. 2006b. Traviando - debugging simulation traces with message sequence charts. In *3rd Int. Conf. on Quantitative Evaluation of Systems*, 135–136: IEEE.

Lang, T., P. Branch, and G. Armitage. 2004. A synthetic traffic model for quake3. In *ACE '04: Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, 233–238. New York, NY, USA: ACM.

Nevill-Manning, C. 1996, May. *Inferring sequential structure*. Ph. D. thesis, University of Waikato.

Nevill-Manning, C. G., and I. H. Witten. 1997. Identifying hierarchical structure in sequences: A linear-time algorithm. *J. Artif. Intell. Res. (JAIR)* 7:67–82.

Wietholter, S., M. Emmelmann, C. Hoene, and A. Wolisz. 2006. Tkn edca model for ns-2. Technical report, Technische Universitat Berlin.

Zapotoczky, J., and K. Wolter. 2008. Increasing performance of the 802.11e protocol through access category shifting. In *Proc. 14th Conf. Measuring, Modelling and Evaluation of Computer and Communication Systems*: VDE-Verlag.

AUTHOR BIOGRAPHIES

NATHAN J. SCHMIDT is a graduate student in the Department of Computer Science at the College of William and Mary. His research interests include networking, simulation, and model checking and debugging techniques. His email address is <njschm@cs.wm.edu>.

PETER KEMPER is an associate professor in the Department of Computer Science at the College of William and Mary (previously Universität Dortmund and TU Dresden, Germany). His research interests include modeling techniques and tools for performance, performability and dependability analysis of systems. He contributed to analysis techniques for the numerical analysis of Markov chains, model checking stochastic models, techniques for simulation optimization. His web page can be found via <www.cs.wm.edu/~kemper> and his email address is <kemper@cs,wm.edu>.