# Investigating Optimal Progress Measures for Verification of the WebSocket Protocol

Lars M. Kristensen - Email: lmkr@hib.no

Department of Computing, Mathematics, and Physics, Bergen University College

## Abstract

The sweep-line method is a state space reduction technique for memory-efficient on-the-fly verification of concurrent systems. The method relies on a progress measure capturing inherent progress in the system under verification to store only fragments of the state space in memory at a time and thereby reduce peak memory usage. The sweep-line method has been applied to many concurrent systems, but the optimality of progress measures in terms of the peak number of states stored has not been investigated. Assessing the optimality of a progress measure is important since memory in most cases is the limiting factor in verification using state spaces. We derive lower bounds for the peak number states and present initial experimental results on near optimal progress measures for verification of the IETF WebSocket protocol.

## 1 Introduction

Most software systems today can be characterised as concurrent systems in that their operation fundamentally relies on communication and synchronisation between concurrently executing and independently scheduled software components. This includes software and protocols for Internet and web-based services, multi-threaded applications, and software for embedded- and networked control systems. It is well-known that the design, test, debugging, and implementation of concurrent systems can be a challenging task. *Model checking* [1] based on state space exploration has emerged as a powerful paradigm for verifying the correctness of finite-state concurrent systems and aid in the development of reliable concurrent software systems. The principle of state space exploration (in its most basic form) is to enumerate all reachable states of the system under verification to algorithmically decide whether a system has certain formally stated behavioural properties or not. The main drawback of state space exploration is the inherent *state explosion problem* [7] which means that for practical verification of systems, reduction techniques coupled with on-the-fly verification of properties need to be employed in order to handle the size of state spaces appearing in typical systems. This has led to the development of a wide collection of state space reduction techniques that exploit different characteristics of the system under verification to reduce the part of the state space that needs to be explored; reduce the amount of memory that needs to be used when exploring the state space; or enable the use of powerful computing infrastructures to increase the CPU and/or memory resources available.

This paper concentrate on the sweep-line state space exploration method [4]. To make the presentation independent of any particular language used for modelling the system under verification, we view state spaces as transition systems $TS = (S, \Delta, s_0)$ consisting of a set of system states $S$, a transition relation $\Delta \subseteq S \times S$, and an initial state $s_0 \in S$. State space exploration consists of exploring the set of *reachable states*, i.e., the set of states that can be reached from the initial state by successively following successor states as determined by the transition relation.

## 2 The Sweep-Line Method and Lower Bounds

The basic idea of the sweep-line method is to exploit a notion of progress inherent in many systems. As examples, progress can be present in control flow, retransmission counters, sequence numbers, and execution phases. A progress measure may be computed from the structure of the model under verification or be provided by the analyst. Progress is captured via a *progress measure* $\phi : S \to O$ that maps each system state $s \in S$ into a progress value $\phi(s)$ belonging to a set of *progress values* $O$. The progress values partition the state space in *progress layers* such that all states in a given layer have the same progress value. The method explores one layer at a time in a least-progress-first order according to a total ordering $\sqsubseteq$ on the set of progress values. The sweep-line method optimistically assumes that the system always makes progress (i.e., successors of a state have equal or larger progress values) and once a layer has been processed, the states of this layer are deleted from memory and the method proceeds to states in layers with a higher progress value. This means that the peak memory usage is reduced compared to ordinary state space exploration where all encountered states are stored in memory. If the system does make regress (i.e., the progress measure is non-monotonic), then states at the end of regress edges are marked as *persistent* and a new search is initiated from such states meaning that some reachable states may be visited several times. A state marked as persistent will not be deleted from memory which ensures termination [4].

The operation of the sweep-line method is specified in Alg. 1. The algorithm starts the exploration (line 2) by marking the initial state $s_0$ as non-persistent, and inserting it into the set of root states $\mathcal{R}$ for the search (exploration) and into the set of states $\mathcal{H}$ that has been encountered and is currently stored in memory. The algorithm then initialises the priority queue of unprocessed states $\mathcal{Q}$ to the set of roots and explores the states reachable from these roots (line 4). In each iteration, states are processed in a least-progress-first order (lines 8-10) and when a progress layer has been processed, non-persistent states in $\mathcal{G}$ are deleted from memory (line

**Algorithm 1** The sweep-line state space exploration algorithm.

```
 1: algorithm Sweep is
 2:    $s_0.pers$ := false ; $\mathcal{R} := \{s_0\}$ ; $\mathcal{H} := \{s_0\}$
 3:    while $\mathcal{R} \neq \emptyset$ do
 4:       $\mathcal{Q} := \mathcal{R}$ ; $\mathcal{R} := \emptyset$ ; $explore()$
 5: procedure explore() is
 6:    while $\mathcal{Q} \neq \emptyset$ do
 7:       $\mathcal{G} := \emptyset$ ; $\phi_m := \mathcal{Q}.minProgress()$
 8:       while $\mathcal{Q}.minProgress() = \phi_m$ do
 9:          $s := \mathcal{Q}.dequeue()$
10:          $process(s)$
11:       $\mathcal{H} := \mathcal{H} \setminus \mathcal{G}$
12: procedure process(s) is
13:    for $(s, s') \in \Delta$ do
14:       if $s' \notin \mathcal{H}$ then
15:          $s'.pers := \phi(s') \sqsubset \phi(s)$
16:          $\mathcal{H} := \mathcal{H} \cup \{s'\}$
17:          if $s'.pers$ then
18:             $\mathcal{R} := \mathcal{R} \cup \{s'\}$
19:          else
20:             $\mathcal{Q} := \mathcal{Q} \cup \{s'\}$
21:    if $\neg s.pers$ then
22:       $\mathcal{G} := \mathcal{G} \cup \{s\}$
```

11). The processing of a state $s$ (lines 12-22) explores all successor states of $s$, and insert any new states in $\mathcal{H}$ and in $\mathcal{R}$ (if persistent) or in $\mathcal{Q}$ (if not persistent). Non-persistent processed states are added to $\mathcal{G}$ (line 22) to ensure that they are deleted (line 11) when the complete progress layer has been processed.

With a pure *internal memory implementation* of the sweep-line method, only the states in the current layer, states in the priority queue of unprocessed states, and states marked as persistent are stored in memory. The key observation to derive a lower bound on the peak number of states stored in the case of monotonic progress measures (i.e., no regress edges and hence no persistent states) is that all states on a cycle of the state space must belong to the same layer. This implies that all states belonging to a strongly connected component *scc* must be present in memory simultaneously. Furthermore, any successor states of states in *scc* with a larger progress value must be stored in the queue of unprocessed states. Using the hybrid *external-memory implementation* presented in [2], states in the queue having a larger progress value can be stored in external memory.

**Theorem 1** *Let SCC denote the set of strongly connected components (SCCs) for a state space* $TS = (S, \Delta, s_0)$*, and for an SCC* $scc \in SCC$*, let* $Out(scc) = \{s' \in S - scc \mid \exists s \in scc : (s, s') \in \Delta\}$*. Then,* $\max_{scc \in SCC} |scc| + |Out(scc)|$ *is a lower bound for the peak number of states stored with the internal memory implementation, and* $\max_{scc \in SCC} |scc|$ *is a lower bound for the hybrid external-memory implementation.*

# 3 Progress Measure for the WebSocket Protocol

In earlier work [6], we conducted formal modelling of the WebSocket (WS) protocol [3] using Coloured Petri Nets (CPNs) and CPN Tools [5]. The WS protocol has been developed by the IETF to transform an HTTP connection into a message-oriented bidirectional connection eliminating the HTTP request-response pattern. As part of our work, we used full state space exploration to verify connection establishment properties of the WS protocol. This led to the identification of potential synchronisation errors in the closing of WS connections, and highlighted the need for reduction techniques to verify larger configurations of the WS protocol.

Figure 1 shows a selected module from the CPN model of the WS protocol, modelling the client-side of the WS connection establishment. From the structure of this module, it is evident that there is progress present in the WS protocol as both the client and the server protocol entities progress through states when going from an IDLE state to an OPEN state to a CLOSED state. This motivated the application of the sweep-line method to the WS protocol.

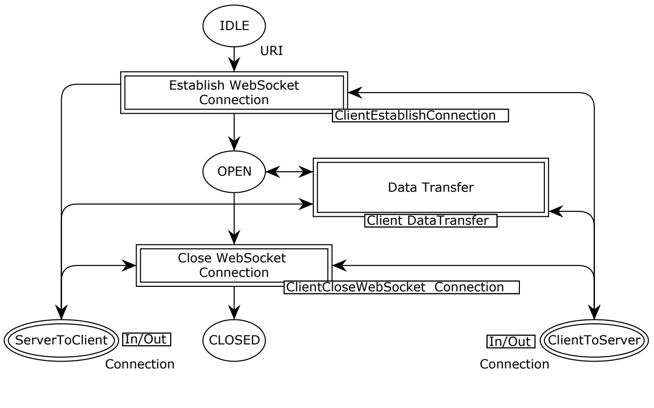
IDLE
URI
Establish WebSocket Connection
ClientEstablishConnection
OPEN
Data Transfer
Client DataTransfer
Close WebSocket Connection
ClientCloseWebSocket_Connection
ServerToClient
In/Out
Connection
CLOSED
In/Out
ClientToServer
Connection

Figure 1: Client module of the WS CPN model.

Table 1 provides selected initial experimental results obtained using CPN Tools for three configurations C1-C3 of the WS protocol. The results are based on a state-based progress measure for the client and server entities as outlined above,

| Config | Peak-In | $\Omega(Peak - In)$ | Peak-Ext | $\Omega(Peak - Ext)$ |
|---|---|---|---|---|
| C1 | 0.47 | 1.02 | 0.32 | 1.00 |
| C2 | 0.56 | 1.01 | 0.38 | 1.00 |
| C3 | 0.64 | 1.01 | 0.45 | 1.03 |

Table 1: Initial Experimental results for WS progress measures.

and additionally considering intermediate sub-states within the three main phases shown in Fig. 1. Column Peak-In lists the peak number of states stored with a pure internal memory implementation relative to the total number of states in the state space, and column $\Omega(Peak - In)$ lists the peak number of states stored relative to the lower bound. Columns Peak-Ext and $\Omega(Peak - Ext)$ give the corresponding numbers for the external-memory implementation. Computation of the lower bounds from Thm. 1 relies on Tarjan's algorithm for computing SCCs. It can be seen that the peak number of states stored ranges from 32% to 64% of the full state space, and that the peak number of states stored is at most 3% larger than the lower bound.

## 4 Outlook

We have presented initial work on lower bounds for the peak number of states stored with the sweep-line in the case of monotonic progress measures. The lower bounds can be efficiently computed using Tarjan's algorithm for small configurations of a system under verification and used to experimentally assess the optimality of a progress measure. Furthermore, we have shown how to define near optimal progress measures for the WS protocol. Future work includes investigating bounds for the non-monotonic case and investigate how tight our bounds are for other examples as we may have multiple SCCs within a progress layer. Future work also includes assessing the progress measures that have been applied for verification of e.g., the DCCP, IOTP, and WAP protocols and possibly suggest better progress measures.

## References

[1] E. M. Clarke, E. A. Emerson, and J. Sifakis. Model Checking: Algorithmic Verification and Debugging. *Commun. ACM*, 52(11):74–84, 2009.

[2] S. Evangelista and L. M. Kristensen. Combining the Sweep-Line Method with the use of an External-memory Priority Queue. In *Proc. of SPIN'12 Symp. on Model Checking of Software*, volume 7385 of *LNCS*, pages 43–61. Springer, 2012.

[3] I. Fette and A. Melnikov. The WebSocket Protocol. tools.ietf.org/html/rfc6455.

[4] K. Jensen, L.M. Kristensen, and T. Mailund. The Sweep-line State Space Exploration Method. *Theoretical Computer Science*, 429:169–179, 2012.

[5] K. Jensen, L.M. Kristensen, and L. Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. *STTT*, 9(3-4), 2007.

[6] K. Simonsen and L.M. Kristensen. Towards a CPN-based Modelling Approach for Reconciling Verification and Implementation of Protocol Models. In *Proc. of MOMPES'12*, volume 7706 of *LNCS*, pages 106–125. Springer, 2013.

[7] A. Valmari. The State Explosion Problem. volume 1491 of *LNCS*, pages 429–528, 1998.