Journal of Computer and System Sciences 82 (2016) 310–332

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Synchronous counting and computational algorithm design

Danny Dolev[a], Keijo Heljanko[b], Matti Järvisalo[c], Janne H. Korhonen[c], Christoph Lenzen[d], Joel Rybicki[b,*], Jukka Suomela[b], Siert Wieringa[b]

[a] The Rachel and Selim Benin School of Engineering and Computer Science, Edmond J. Safra Campus, The Hebrew University of Jerusalem, Israel
[b] Helsinki Institute for Information Technology HIIT, Department of Computer Science, Aalto University, Finland
[c] Helsinki Institute for Information Technology HIIT, Department of Computer Science, University of Helsinki, Finland
[d] Department of Algorithms and Complexity, MPI for Informatics, Saarbrücken, Germany

ARTICLE INFO

Article history:
Received 8 January 2015
Received in revised form 1 September 2015
Accepted 11 September 2015
Available online 23 October 2015

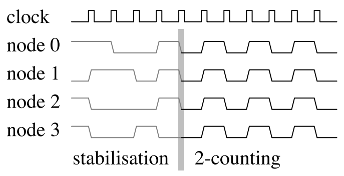
Keywords:
Distributed computing
Self-stabilisation
Byzantine fault tolerance
Synthesis
Formal methods
SAT

ABSTRACT

Consider a complete communication network on n nodes. In *synchronous* 2*-counting*, the nodes receive a common clock pulse and they have to agree on which pulses are "odd" and which are "even". Furthermore, the solution needs to be *self-stabilising* (reaching correct operation from any initial state) and tolerate f *Byzantine failures* (nodes that send arbitrary misinformation). Prior algorithms either require a source of random bits or a large number of states per node. In this work, we give fast state-optimal deterministic algorithms for the first non-trivial case $f = 1$. To obtain these algorithms, we develop and evaluate two different techniques for algorithm synthesis. Both are based on casting the synthesis problem as a propositional satisfiability (SAT) problem; a direct encoding is efficient for synthesising time-optimal algorithms, while an approach based on counter-example guided abstraction refinement discovers non-optimal algorithms quickly.

1. Introduction

Synchronous counting In the *synchronous C-counting* problem, n nodes have to count clock pulses modulo C. Starting from any initial configuration, the system has to *stabilise* so that all nodes agree on the counter value. Put otherwise, eventually all nodes have to consistently label each clock pulse with values incrementing modulo C.



In this work, we consider a fully-connected synchronous communication network of n nodes with identifiers from the set $\{0, 1, \ldots, n-1\}$. Each node is a finite state machine with s states, and after every state transition, each node *broadcasts* its current state to all other nodes—effectively, each node can see the current states of all other nodes. An algorithm specifies (1) the new state for each observed state, and (2) how to map the internal state of a node to its output.

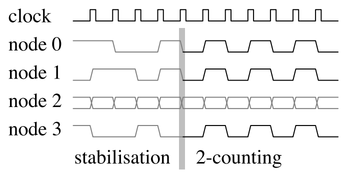
* Corresponding author.
E-mail address: joel.rybicki@aalto.fi (J. Rybicki).

http://dx.doi.org/10.1016/j.jcss.2015.09.002
0022-0000/© 2015 Elsevier Inc. All rights reserved.

Byzantine fault tolerance In a fault-free system, the C-counting problem is trivial to solve. For example, we can designate node 0 as a leader, and then all nodes (including the leader itself) can follow the leader: if the current state of the leader is c, the new state is $c + 1 \bmod C$. This algorithm will stabilise in time $t = 1$, and we only need $s = C$ different states.

However, we are interested in algorithms that tolerate *Byzantine failures*. Some number f of the nodes may be *faulty*. A faulty node may send arbitrary misinformation to non-faulty nodes, including *different* information to different nodes within the same round. For example, if we have nodes $0, 1, 2, 3$ and node 2 is faulty, node 0 might observe the state vector $(0, 1, 1, 1)$, while node 1 might observe the state vector $(0, 1, 0, 1)$.

Our goal is to design an algorithm with the following guarantee: even if we have up to f faulty nodes, no matter what the faulty nodes do, the system will stabilise so that after t rounds all non-faulty nodes start to count clock pulses consistently modulo C. We will give a formal problem definition in Section 4.



Synchronous counting can be used as a fault-tolerant co-ordination primitive in systems where a synchronous clock signal is available, but the clock pulses have not been labelled in any manner, for example, there is no distinction between even and odd clock pulses. In general, a C-counter can be used as a fault-tolerant *round counter* that assigns explicit round numbers for each clock pulse.

State of the art Both randomised and deterministic algorithms for synchronous counting (often also referred to as *digital clock synchronisation*) have been presented in the literature (see Section 2). However, prior algorithms tend to be expensive to implement in hardware: they require a source of random bits or complicated circuitry.

In this work, we use a single parameter s, the number of states per node, to capture the complexity of an algorithm. If one resorts to randomness, it is possible to solve 2-counting with the trivially optimal number of $s = 2$ states—at the cost of a slow stabilisation time (see Sections 2 and 5). However, it is not at all clear whether a small number of states suffices for *deterministic* algorithms.

Contributions We employ *computational* techniques to design deterministic 2-counting algorithms that have the smallest possible number of states. Our contributions are two-fold:

1. we present new algorithms for the synchronous counting problem,
2. we develop new computational techniques for constructing self-stabilising Byzantine fault-tolerant algorithms.

Our focus is on the first non-trivial case of $f = 1$. The case of $n = 1$ is trivial, and by prior work it is known that there is no algorithm for $1 < n < 4$. We give a detailed analysis of 2-counting for $n \geq 4$:

- there is no deterministic algorithm for $f = 1$ and $n = 4$ with $s = 2$ states,
- there is a deterministic algorithm for $f = 1$ and $n \geq 4$ with $s = 3$ states,
- there is a deterministic algorithm for $f = 1$ and $n \geq 6$ with $s = 2$ states.

Overall, we develop more than a dozen different algorithms with different characteristics, each of which can be also generalised to a larger number of nodes. See Fig. 1 for an overview of the time–space tradeoffs that we achieve with our algorithms.

With very few states per node, our algorithms are easy to implement in hardware. For example, a straightforward implementation of our algorithm for $f = 1$, $n = 4$, and $s = 3$ requires just 2 bits of storage per node, and a lookup table with $3^4 = 81$ entries. All of our computer-designed algorithms are freely available online [1] in a machine-readable format. While our algorithms are synchronous 2-counters, they can be easily composed to construct synchronous 2^b-counters for any positive integer b (see Section 3 for details).

This work can be seen as a case study of applying synthesis techniques in the area of distributed algorithms. We demonstrate that the synthesis of non-trivial self-stabilising Byzantine fault-tolerant algorithms is indeed possible with the help of modern propositional satisfiability (SAT) solvers [6,26]. We describe two complementary approaches for the synthesis of synchronous 2-counting algorithms and give an empirical comparison of their relative performance:

1. a direct encoding as SAT,
2. a SAT-based counter-example guided abstraction refinement (CEGAR) [13,14] approach.