BIROn - Birkbeck Institutional Research Online

Zhou, Y. and Zhou, N. and Han, Tingting and Gu, J. and Wu, W. (2018) Probabilistic verification of hierarchical leader election protocols in dynamic systems. Frontiers of Computer Science 12 (4), pp. 763-776. ISSN 2095-2228.

Downloaded from: https://eprints.bbk.ac.uk/id/eprint/19777/

Usage Guidelines:
Please refer to usage guidelines at https://eprints.bbk.ac.uk/policies.html or alternatively contact lib-eprints@bbk.ac.uk.

Front.Comput.Sci.
DOI

RESEARCH ARTICLE

Probabilistic Verification of Hierarchical Leader Election Protocol in Dynamic Systems

Yu Zhou(✉)[1], Nvqi Zhou [1], Tingting Han [2], Jiayi Gu [1], Weigang Wu [3]

[1] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, 210016, China
[2] Department of Computer Science and Information Systems, Birkbeck, University of London, UK
[3] Department of Computer Science, Sun Yat-sen University, 510006, Guangzhou, China

© Higher Education Press and Springer-Verlag Berlin Heidelberg 2015

Abstract Leader election protocols are fundamental for coordination problems—such as consensus—in distributed computing. Recently, hierarchical leader election protocols have been proposed for dynamic systems where processes can dynamically join and leave, and no process has global information. However, quantitative analysis of such protocols is generally lacking. In this paper, we present a probabilistic model checking based approach to verify quantitative properties of these protocols. Particularly, we employ the compositional technique in the style of assume-guarantee reasoning such that the sub-protocols for each of the two layers are verified separately and the correctness of the whole protocol is guaranteed by the assume-guarantee rules. Moreover, within this framework we also augment the proposed model with additional features such as rewards. This allows the analysis of time or energy consumption of the protocol. Experiments have been conducted to demonstrate the effectiveness of our approach.

Keywords distributed computing, hierarchical leader election protocol, dynamic systems, probabilistic model checking

1 Introduction

Eventual leader election protocols are fundamental to solve coordination related problems in distributed computing [1]. Generally speaking, in the *leader election* problem, there are n processes in the system, each of which has a unique identity. Upon termination of the protocol, exactly one process announces itself as the leader [2]. A number of leader election protocols have been designed for various kinds of settings [3–5]. Recently, with the rapid development of networking technology, *dynamic systems* are becoming increasingly popular. Different from the traditional assumption made in the static system, in dynamic systems, processes can dynamically join and leave the system, and no process has the global information on the whole [6]. Indeed, many applications exhibit the features of the dynamic system, and typical examples include Peer-to-Peer systems, Internet-of-Things systems, etc. Therefore, *eventual leader election* in dynamic systems has been intensively studied in the literature [7–9]. The main challenge here is the inherent uncertainty of the underlying dynamic models. This uncertainty, not only brings difficulty to the protocol design, but also to the (quantitative) analysis of the proposed protocols. Indeed, in [9], we gave a theoretical proof of the protocol's correctness. However, such correctness cannot answer questions such as how many steps are needed, or how much energy is consumed, to elect a global leader on average. Such quantitative analysis is evidently of great interest to the end users of the protocol.

Model checking turns out to be an effective formal method to verify a design artifact against certain properties, i.e., checking whether a design is a model of the specification. Normally, the design model is expressed as a labeled transition system and the property is written in temporal logics. Because the verification is conducted automatically and exhaustively, and a counterexample can be provided when the specification is unsatisfied, model checking has been widely applied in practice [10]. However, conventional model checking techniques can only give a *yes* or *no* answer to the

Received month dd, yyyy; accepted month dd, yyyy

E-mail: *zhouyu@nuaa.edu.cn*

properties in question — it cannot analyze systems which exhibit probabilistic behavior in a satisfactory way. Probabilistic model checking generalizes conventional model checking in that it builds and analyzes probabilistic models such as Markov chains and Markov decision processes [11]. By leveraging probabilistic extensions of temporal logics, the quantitative properties of these models can be specified or verified automatically.

In light of these considerations, we adopt probabilistic model checking to quantitatively verify properties — related to the scalability and efficiency issues — of a hierarchical leader election protocol for dynamic systems which was recently proposed in the literature [9]. Particularly, we leverage the PRISM model checker [12] to construct the hierarchical model. PRISM is a leading open-source model checker and has been applied in many fields, such as communication protocols, distributed algorithms and some other systems of specific subjects like biology. The paper is based on our previous work on the design of hierarchical eventual leader election protocols for dynamic systems [9] and probabilistic analysis of regular leader election protocols without cluster hierarchies [13]. In this paper, we make the following contributions:

1. We complement the analysis of our model proposed in [9] with quantitative verification via probabilistic model checking. To our best knowledge, this is the first case study of applying probabilistic verification on a hierarchical eventual leader election protocol. The verification gives deeper insights on the properties of the protocol.
2. We adopt a compositional reasoning technique, i.e., assume-guarantee, to verify hierarchical protocol design, which gives better scalability compared to the holistic approach.
3. We extended the original protocol model in two aspects. Firstly, we employ cost/rewards to model energy consumption; secondly, we relax the assumption of reliable communication. For both aspects, we use probabilistic model checking to conduct quantitative analysis.

Moreover, as a minor contribution, we develop some techniques to increase the expressiveness of PRISM model checker and an automated code generation technique to increase the scales of models. These techniques can be reused as a library for other similar problems. The rest of the paper is structured as follows: Section 2 briefly introduces the hierarchical leader election protocol in dynamic systems. The detailed model design of the protocol is described in Section 3 followed by the probabilistic verification in Section 4. Some discussions and related work are given in Section 5 and Section 6 respectively. Section 7 concludes our paper.

2 Hierarchical Leader Election Protocol in a Nutshell

In this section, we first briefly introduce the hierarchical design of the protocol and assumptions of the model. We then present some essential parts of the protocol. The details of the protocol can be found in [9].

The hierarchical leader election protocol is designed for the cluster based hierarchy topology, which has been widely used in many fields such as consensus [14] and information dissemination [9, 15]. Generally, the hierarchical election of an eventual leader is achieved in two steps. In the fist step, election is conducted to select a cluster head within a cluster. In the second step, the election is conducted among cluster heads which have been done in the first step to select the globally unique leader. Correspondingly, the protocol consists of two layers, i.e., the lower layer and the upper layer. In the lower layer, cluster heads are elected with each cluster, and then in the upper layer, election is conducted among cluster heads so as to elect the eventual leader of the whole system. Both layers adopt the *query-response* communication primitives suited to the dynamic system. In a nutshell, there are two communication primitives: *broadcast* and *wait until*. The former is used to broadcast a query message to all of the processes; the latter stipulates that the process waits until a pre-defined number α of responses have been received. Here, α defines the minimal number of stable processes; a process is stable if it never leaves a cluster after joining the cluster. It captures the progress requirement of the dynamic system [6], and normally we assume $\alpha > \lfloor n/2 \rfloor + 1$. The communication channel is assumed to be reliable in the original work for simplicity [9], but we shall relax this assumption to a more realistic setting during the quantitative verification in our model.

The election protocol in the *lower layer* is enacted within each individual cluster and the purpose is to select the head of the corresponding cluster. The *upper layer* protocol stands on the whole system and its task is to guarantee a unique leader of the system to be elected. In short, the protocol runs sequentially: the cluster head is elected in the lower layer, and then the eventual leader is elected among those cluster heads in the upper layer.

Lower layer protocol. The original protocol in this layer consists of four tasks, i.e., *Task 1, Task 2, Task 3* and *Task 4*. These four tasks are executed independently by the processes in the lower layer. These tasks are interacted and synchronized through message exchanges. Among them, *Task 1* is the core of the protocol. It is used for processes to exchange information and update cluster-head candidates. The main purpose is to reduce the size of the *candidate head set* denoted by $trust_i$ and to ensure a unique cluster head is elected eventually.

```
begin
    Init: $rec\_from_i = \Pi$; $log\_date_i = 0$;
    $trust_i = \Pi$; $leader_i = i$; $CH = i$;
    $rec_i = \Pi$; $seqnum_i = 0$; $accept_i = \Pi$;
    while true do
        broadcast QUERY(i) to the whole cluster;
        wait until RESPONSE(j,$rec\_from_j$) received from $\alpha$
        processes;
        $RECFROM_i$ = the union of $rec\_from$ of those senders;
        $trust_i = trust_i \cap RECFROM_i$;
        $rec\_from_i$ = the set of the senders which send RESPONSE;
        if $trust_i$ is modified then
            broadcast TRUST($trust_i$,$log\_date_i$) to the whole
            system;
        end
    end
end
```

Algorithm 1: Lower layer protocol: Task 1 (main task)

The data structures used in the lower layer protocol are listed as follows:

- Π is the full set of processes;
- rec_from_i is the set of processes from which process p_i receives a RESPONSE message;
- $trust_i$ is the candidate cluster head set;
- log_date_i is the logical time which defines the age of $trust_i$;
- $leader_i$ is the current global leader process;
- CH_i is the local cluster head of process p_i;
- RECFROM$_i$ is the union set of rec_from

Algorithm 1 illustrates these data structures and their initialization. In *Task 1*, firstly, p_i sends a QUERY message and then keeps waiting until α RESPONSE messages have been received. Next, it updates its rec_from_i set based on the identities of those processes which have sent RESPONSE messages. After that it adjusts RECFROM$_i$ set by computing the union of the rec_from of all above the processes appearing in rec_from_i. Afterwards, $trust_i$ is modified by set intersection. If the value of $trust_i$ is changed, then p_i broadcasts another message containing the TRUST set to the whole cluster so that all processes in the cluster can adjust their leader information accordingly.

Task 2 shown in Algorithm 2 is to ask a process to send a RESPONSE message immediately after it receives a QUERY message from other processes. We note that, in the original protocol, it is assumed that the channel for message delivery is reliable and thus no message could lost. This is, however, an unrealistic assumption in the real networking environment. To remedy this, we introduce a parameter *ratio_suc* to represent the probability of successfully sending messages.

```
begin
    upon QUERY($i$) is received from $p_i$ $ratio\_suc$: send
    RESPONSE($j$, $rec\_from_j$) to $p_i$;
end
```

Algorithm 2: Lower layer protocol: Task 2 (send back RESPONSE)

Task 3 is designed to modify the *trust* set by comparing to log_date_i when receiving the TRUST messages. At the same time, this task can also monitor the *trust* set and reset it if it is empty. Algorithm 3 illustrates *Task 3*. Once p_i receives a TRUST message from process p_j, it should adjust its $trust_i$ set accordingly. *Task 4* is illustrated in Algorithm 4 and is used to update the cluster head of a process when it has finished executing Task 3, i.e., the process with the smallest identity in the candidate set will be announced as the cluster head.

```
begin
    upon TRUST($trust_i$, $log\_date_i$) is received from $p_i$ if
    $log\_date_i == log\_date_j$ then
        $trust_j = trust_j \cap trust_i$;
    end
    if $log\_date_i > log\_date_j$ then
        $trust_j = trust_i$;
        $log\_date_j = log\_date_i$;
    end
    if $trust_j == \emptyset$ then
        $trust_j = \Pi$; $log\_date_j = log\_date_j + 1$;
    end
end
```

Algorithm 3: Lower layer protocol: Task 3 (update *trust* set)

```
begin
    if $trust_i == \emptyset$ or $trust_i == \Pi$ then
        $CH_i = i$;
    else
        $CH_i = \min(trust_i)$;
    end
end
```

Algorithm 4: Lower layer protocol: Task 4 (determine the cluster head)

Upper layer protocol. The upper layer of this hierarchy-based leader election protocol is to elect a global leader of the whole system among all cluster heads which have been elected in the lower layer. The following data structures are used:

- $accept_i$ is the set of candidate leaders; $seqnum_i$ is a logical time which defined the age of $accept_i$;
- rec_i is the set of (C_k, k) from which p_i received a RES message;
- CLUSTER$_i$ is the union set of C_k

Two messages are used to exchange information. $ALIVE(C_i, i, accept_i, seqnum_i)$ is a message to gossip with other cluster heads, and $RES(C_k, k, rec_k)$ is the response message of ALIVE.

In this layer, there are three important tasks to elect a global leader for the whole system, i.e., *Task 1, Task 2, and Task 3*. Among them, *Task 1* is the main body of the protocol which decreases the number of the candidate leaders of the system via continuously updating the leader set. *Task 2* is an auxiliary task dealing with the condition where a cluster head receives the ALIVE message. *Task 3* is similar to its counterpart in the lower layer targeting at updating the leader set. The above three tasks are shown with pseudo-code in Algorithm 5-7.

```
begin
    while $true$ do
        if $p_i$ is the head of a cluster $C_i$ then
            broadcast ALIVE($C_i, i, accept_i, seqnum_i$) to the whole
            system;
            wait until enough RES($C_k, k, rec_k$) having been
            received from (n-f) clusters;
            $rec_i$ = the set of the senders which send RES firstly in
            their own clusters;
            Let $CLUSTER_i$ = the union of rec of those senders;
            $accept_i = accept_i \cap CLUSTER_i$;
            if $cluster_j$ has more than one leaders then
                $accept_i = accept_i - \{j\}$;
            end
        end
    end
end
```

Algorithm 5: Upper layer protocol: Task 1 (main task)

Algorithm 5 illustrates *Task 1* of the upper layer protocol. At first, cluster head p_i broadcasts an ALIVE message to all cluster heads in the system, and keeps waiting until $(n - f)$ RES messages from different clusters received, in which n is the number of clusters and f is the number of empty clusters where there are no processes. Next, p_i records the *id*s of each cluster head and sends the values to rec_i. After that, CLUSTER$_i$ is updated by computing the union of *rec* set of cluster heads appearing in rec_i. Then, $accept_i$ is intersected with the above CLUSTER$_i$. Eventually, p_i checks its $accept_i$ to test whether or not a cluster has more than one cluster head and then deletes the *id*s of those clusters.

```
begin
    upon ALIVE($C_i, i, accept_i, seqnum_i$) is received from $p_i$
    if $seqnum_i == seqnum_j$ then
        $accept_j = accept_j \cap accept_i$;
    else if $seqnum_i > seqnum_j$ then
        $accept_j = accept_i$; $seqnum_j = seqnum_i$;
    if $accept_j == \emptyset$ then
        $accept_j = \Pi$; $seqnum_j = seqnum_i$+1;
    end
    if $C_i == C_j \&\& j! = i$ then
        $accept_j = accept_j - \{C_j\}$;
    else
        send RES($C_j, j, rec_j$) to $\Pi$;
    end
end
```

Algorithm 6: Upper layer protocol: Task 2 (update *accept* and send *RES* back)

Algorithm 6 illustrates *Task 2* of the upper layer. In the first place, when p_j, the head of cluster C_j, receives an ALIVE message from p_i, p_j modifies its value after comparing the *seqnum* values of two cluster heads. Next, p_j checks its $accept_j$ value to decide whether or not it is empty. If there is no element in this set, p_j resets $accept_j$ with all elements. And then, p_j will check whether this message is sent from other processes within the same cluster. If it is true, p_j will update the $accept_i$ by removing its cluster id; otherwise, it will send a RES message.

```
begin
    if $accept_i == null$ then
        $leader_i = i$;
    else
        $leader_i$ = the cluster head of min($accept_i$);
    end
end
```

Algorithm 7: Upper layer protocol: Task 3 (determine the leader)

Task 3 of the upper layer protocol finishes the election. This task targets at electing the global leader of the whole system. When p_i finishes a cycle of the operation, it immediately updates its leader information according to the value of $accept_i$. This task is similar to the *Task 4* of the lower layer protocol. The above tasks constitutes the essential steps of the two layer eventual leader election protocol. We refer interested readers to [9] for details.

3 Model of the Protocol

In this section, we give the modeling of the protocol presented in the previous section using PRISM. We assume that readers are familiar with the syntax and features of the PRISM modeling language, for which unfamiliar readers can refer to [12]. PRISM uses a state based language to construct the models, and in this way, it has a natural correspondence to the labeled transition system during model checking. The paradigm of the state based modeling language is quite different from the imperative language we used to describe our protocol. Therefore, the construction of the corresponding PRISM model is not straightforward. On the other hand, as we can see, the hierarchical protocol contains seven tasks in total and if we faithfully modeled the protocol, there would be too many intermediate states obstructing efficient verification seriously. Thus we need to abstract some unnecessary details away in the model design process. Concretely speaking, we change the way of constructing the RECFROM set and simplify the way of dealing with TRUST message.

(1) Constructing the RECFROM. In this part, we show how to reconstruct the RECFROM set via a simple way without changing the essence of the original protocol. From lower layer *Task 1*, we observe that RECFROM is a union of the rec_from sets of α processes having sent RESPONSE messages. These α rec_from sets contain *id*s of the α processes. Based on this, we can redesign some steps of the original protocol. During the period of computing rec_from set, we only need to focus on the size of the set rec_from which is always α — the elements of rec_from are irrelevant. Note that here, it should be guaranteed that $\alpha \geq \lfloor n/2 \rfloor + 1$. Therefore, when the cluster has n processes, there are $\binom{n}{\alpha}$ different possibilities of selecting α processes from such a cluster. In addition, we should pay attention to the construction of RECFROM set. It consists of α rec_from each containing α different elements, and thus the total number is α^2. However, some of these identities are duplicate, and thus the number of non-redundant elements in this set is between α and n. Since there is no difference among processes, for each configuration of RECFROM set, we can use one instance to represent. There are $n - \alpha + 1$ such configurations.

(2) Simplifying the way of dealing with the TRUST message. When $trust_i$ is changed, process p_i immediately broadcasts a TRUST message to other processes in the same cluster to update their own trust sets. Then the related processes compare the value of log_date_i and reacts accordingly. Because in the same cluster, all processes are executing in parallel, the change of log_date information is totally stochastic, and thus we use the nondeterminism provided by PRISM to model it and select one of three operations randomly.

The original protocol [9] has an eventual cluster stability assumption, which states that eventually at most f out of all n clusters are empty, and at most s of the clusters are not empty but with less than α members. It requires that after some time, a stable cluster has no more process joining. Accordingly, in our model, we did not consider the processes joining and leaving across different clusters and only considered the situations in stable clusters.

Below we present the modeling of the two layer's protocol.

Lower layer model. First of all, we introduce some variables to model the lower layer protocol. Their definitions and (intuitive) semantic explanations are summarized in Table 1.

As defined in *Task 1* of the lower layer protocol, process p_i broadcasts a QUERY message in its cluster. After that, p_i needs to update its state. In our model, we use the flag variable QUERY$_i$ to label the state and set its value to be *true* after the transition. Without loss of generality, we use $p1$ as the representative process in the examples of PRISM code throughout the paper. The corresponding PRISM model snippet is given in Listing 1.

Table 1 Variable definition and explanations in lower layer

Variable	Semantics
leader$_i$:[1..n] init 1	the leader of p_i's cluster, initial value is its *id*
QUERY$_i$:bool init false	the flag mark of p_i's QUERY, initial value is *false*
R_{i_j}:bool init true	the j^{th} value of p_i's *trust* set, initial value is *true*
TRUST$_i$:bool init false	the listener of p_i's TRUST set, initial value is *false*
s_i: init 0;	the current step of p_i's execution, initial value is 0

Listing 1 Broadcasting a query

```
[]  (s1=0)&(QUERY1=false)->(s1'=1)&(QUERY1'=true);
```

Afterwards p_i receives a collection of responses RECFROM, whose contents are identities of peer processes. As discussed previously, the range of RECFROM is between α and n. So we can adjust the value of R_{i_j} to denote whether the process p_j is included in the response messages. This is exemplified in Listing 2.

Listing 2 Broadcasting a query

```
[]  (s1=1)&(QUERY1=true)->r1:(s1'=2)
    +r2:(s1'=2)&(R1_1'=false)
    +r3:(s1'=2)&(R1_1'=false&R1_2'=false)
    +r4:(s1'=2)&(R1_1'=false&R1_2'=false&R1_3'=false)
    +  ...
```

Once getting the set of RECFROM, we need to decide whether or not p_i broadcasts the TRUST message to all processes within the same cluster after updating *trust*. In our model, we compare RECFROM and *trust*. If they are the same, then it means the *trust* set will not be updated and neither will the update message be broadcast. This step is illustrated in Listing 3. In the example code of this step, *n* denotes the number of processes involved in the election and *t1_i* represents the element of TRUST set.

Listing 3 Comparison between *RECFROM* and *trust*

```
[]  (s1=2)->(s1'=3)&(TRUST1'=(R1_1=t1_1?false:true)
    |(R1_2=t1_2?false:true)|...|(R1_n=t1_n?false:true));
```

If p_i receives the TRUST message sent from other processes, it will execute the corresponding operations and updates its *trust* set. Different from previous steps, it requires synchronization with other processes, i.e., upon receiving the TRUST message, all processes need to stay synchronized. Therefore, we use a variable *A1* to denote this. The PRISM code snippet is explained in Listing 4.

After this step, we proceed to update the *trust* set and QUERY. If the set is empty, then we reset it with its initial value. QUERY is set to be false so as to enable the repetition. This is illustrated in Listing 5.

Listing 4 Synchronizing the processes

```
//process p1
[A1]  (s1=4)&(TRUST1=true)->(s1'=5)&(TRUST1'=false);

//process p2,...,pn
[A1](s1=4)&(TRUST1=true)->
   1/3:(s2'=5)&(t2_1'=t2_1&t1_1)
   &...&(t2_n'=t2_n&t1_n)
   +1/3:(s2'=5)&(t2_1'=t1_1)&...&(t2_n'=t1_n)
   +1/3:(s2'=5);
```

Listing 5 Resetting the *trust* set and *query*

```
[](s1=5)&(t1_1=false)&...&(t1_n=false)->(t1_1'=true)
                                 &...&(t1_n'=true)&(s1'=6);
[](s1=5)&!((t1_1=false)&...&(t1_n=false))->(s1'=6);
[](s1=6)->(QUERY'=false)&(s1'=0);
```

In order to select a leader of a certain cluster in the lower layer, we set up a listener globally to update the leader information of each involved process. When the QUERY information is updated, it will trigger the listener. Therefore, we use the synchronization utility in PRISM to implement the listener as illustrated in Listing 6.

Listing 6 Resetting the *trust* set and *query*

```
[A2](s1=6)->(leader1'=(t1_1=true?1:
              (t1_2=true ?2:(...(t1_(n-1)=true ?(n-1):n)))));
[A2](s1=6)->(QUERY'=false)&(s1' = 0);
```

Upper layer model. Similarly, we first introduce additional variables defined to facilitate the model design, and then we present the essential steps for the upper layer protocol. We also use p_1 as an example. p_1 has a variable, i.e., *Leader*1, to record the head information of its cluster, a Boolean variable ALIVE1 to denote whether it has broadcast the ALIVE message to all cluster heads within the system. The sets of *accept* and *CLUSTER* are the same as defined in the upper layer protocol. These variables and explanations are given in Table 2. The number of cluster heads are denoted as a variable *m*.

Table 2 Variable definitions and explanations in upper layer

Variable	Semantics
*Leader*1*:[0..m] init 1*	the leader of p_1's cluster, initial value is its *id*
*ALIVE*1*:bool init false*	the boolean flag mark of p_i , whether or not p_i sends an alive message, initialized *false*
$accept1_j$:[0..1] init 1	the j^{th} value of p_1's *accept* set, initial value is *0*
s_1*: init 0*	the current step of p_i's execution, initial value is 0

Firstly, p_1 needs to broadcast an ALIVE message to all processes within the system. we model this step and change its states afterwards as illustrated in Listing 7.

Listing 7 Broadcasting an ALIVE message

```
[](s1=0)&(ALIVE1=false)->(s1'=1)&(ALIVE1'=true);
```

After sending the ALIVE message, p_1 and other processes need to synchronize their reactions accordingly. For p_1, it just transfers to the next state; while for other processes, they need to start *Task 2* of the upper layer protocol to handle the ALIVE2 message. In the Listing 8, we use process p_2 as a representative of other processes. The $r1$, $r2$ and $r3$ are the probability coefficients. The sum of the three values should be equal to 1.

Listing 8 Handling the ALIVE message

```
//For p1;
[ali](s1=1)&(ALIVE1=true)->(s1'=2);
...
//For other processes, e.g. p2;
[ali](s1=1)&(alive1=true)->
      r1:(accept2_1'=(accept2_1=accept1_1)?accept2_1:0)&...
      &(accept2_m'=(accept2_m=accept1_m)?accept2_m:0)
      &(s2'=2)+r2:(accept2_1'=accept1_1)&...
      &(accept2_m'=accept1_m)&(s2'=2)+r3:(s2'=2);
```

When p_1 finishes the above operation, it will check the emptiness of its *accept* set. If the set is empty, it will reset with its initial value. Otherwise, it will move to the next state. This step is illustrated in Listing 9.

Listing 9 Resetting the *accept* set

```
[s](s1=2)&(accept1_1=0)&...&(accept1_m=0)->
    (accept1_1=1)&...&(accept1_6=1)&(s1'=3);
[s](s1=2)&!((accept1_1=0)&(accept1_2=0)
      &...&(acccept1_m=0))->(s1'=3);
```

Listing 10 Calculating the *CLUSTER* set

```
[](s1=4)->(s1'=5)&(accept1_1'=
  (accept1_1=1&cluster1_1=1)?accept1_1:0)
  &...&(accept1_m'=(accept1_m=1
 &cluster1_m=1)?accept1_m:0);
```

Process p_1 will get a *CLUSTER* set after broadcasting the *ALIVE* message. We use the similar strategy of calculating RECFROM in the lower layer protocol to get *CLUSTER*. The code snippet is given in Listing 10.

The above steps constitute the essential parts of the upper layer leader election model. However, we also need an external listener to update the leader information after each election cycle. This step is similar to the one in the lower layer protocol and is illustrated in Listing 11, where m denotes the number of cluster heads participating the upper layer election.

Listing 11 Calculating the *CLUSTER* set

```
[A2](s1=5)->(Leader1'=(accept1_1=0&accept1_2=0
          &...&accept1_m=0)?1:(accept1_1=1?1:
          (accept1_2=1?2:...(accept1_(m-1)=1?(m-1):m)...)));
```

4 Probabilistic Verification

4.1 Assume-guarantee based compositional reasoning

Assume-guarantee verification is a frequently used technique in compositional reasoning. It can be used to verify the system S, which consists of the parallel composition of two subsystems S_1 and S_2, written as $S_1||S_2$, against certain property G, written as <*true*>$S_1||S_2$<G>. In order to verify this assertion, it is reduced to verify whether or not the two assertions <*true*>S_1<A> and <A>S_2<G> hold, where A is the assumption and G is the guarantee. Assume-guarantee is a successful compositional verification technique and has broad applications in model checking.

In [16, 17], Kwiatkowska et al. adapted the assume-guarantee technique to probabilistic verification, which means the assumption can also include quantitative properties. In our case, the hierarchical eventual leader election protocol is inherently a layered architecture and the whole election process consists of two sequential sub processes, i.e., election within a cluster, and election within cluster heads. This fact implies the opportunity of leveraging compositional verification technique to tackle the complexity. From the description of the above protocol, we observe the precondition of the upper layer election is that the lower layer cluster can elect a unique head. Therefore, we introduce the assumption that the model of lower layer can eventually elect a unique head within a cluster.

As a result, the rules proposed in [16,17] can be applied in our setting. The reason is that the focus therein is the parallel composition of different modules, while here we need to deal with the sequential composition, for which a new rule is required. Let's start with some basic definitions. We write $\mathcal{D}(S)$ for the set of probabilistic distributions over S.

Definition 1 (Probabilistic automata)**.** *A probabilistic automaton (PA) is a tuple* $T = (S, \alpha, A, \Delta, L)$ *where*

- *S is a set of states;*
- *α is the initial distribution;*
- *A is a set of actions;*
- *$\Delta \subseteq S \times A \times \mathcal{D}(S)$ is a probabilistic transition relation;*
- *$L : S \to 2^{AP}$ is a labeling function.*

Intuitively, at any state s of a PA T, a transition $s \xrightarrow{a} \mu$, where $a \in A$ is an action label and μ is a discrete probability distribution over S, is available if $(s, a, \mu) \in \Delta$. In an execution of the model, the choice between the available transitions at each state is nondeterministic; the choice of successor state is then made randomly according to the distribution μ.

We define a sequential composition operator $\circ$ over two probabilistic automata, $T_1 = (S_1, \alpha_1, A_1, \Delta_1, L_1)$ and $T_2 = (S_2, \alpha_2, A_2, \Delta_2, L_2)$ where $A_1 \cap A_2 = \emptyset$ and $S_1 \cap S_2 = \emptyset$. Note that these conditions can be easily satisfied by renaming. The operator $\circ$ is parameterized by (F, π) where

- $F \subseteq S_1$ is a subset of *absorbing* states of T_1.
- $\pi : F \to S_2$ is a *total* function from F to S_2.

For the sake of simplicity, when applying the operator $\circ$, we assume tacitly the associated F, π are given.

Formally, $T_1 \circ T_2$ is given as

$$(S, \alpha, A, \Delta, L)$$

where

- $S = S_1 \uplus S_2$, i.e., the disjoint union of S_1 and S_2;
- $\alpha = \alpha_1$;
- $A = A_1 \uplus A_2$;
- $\Delta = \Delta_1 \upharpoonright_{S_1 \setminus F} \cup \Delta_2 \cup \{(s, -, \delta_{\pi(s)}) \mid s \in F\}$; and
- $L(s) = \begin{cases} L_1(s) & \text{if } s \in S_1 \\ L_2(s) & \text{if } s \in S_2 \end{cases}$

Note here $\Delta_1 \upharpoonright_{S_1 \setminus F} := \{(s, a, \mu) \in \Delta_1 \mid s \in S_1 \setminus F\}$, $\delta_{\pi(s)}$ is the Dirac distribution, and $-$ means the action name is irrelevant here.

It is straightforward to verify that $T_1 \circ T_2$ is indeed a PA. We introduce the following assume-guarantee rule, specially for the reachability property. Admittedly this is limited comparing to the one in [16, 17] which addresses general safety fragment of PCTL. However, it is sufficient for the purpose of the current paper.

Recall that $T \models [\Diamond F]_{\geq p}$ if for all schedulers of T, the probability of reaching F is no less than p.

Proposition 2. *The following rule holds:*

$$\frac{< true > T_1 < [\Diamond F_1]_{\geq p_1} > \quad < \alpha(\pi(F_1)) = 1 > T_2 < [\Diamond F_1]_{\geq p_2} >}{< true > T_1 \circ T_2 < [\Diamond F_1]_{\geq p_1 \cdot p_2} >}$$

The correctness of the rule is a direct consequence of the Markov property and the definitions and hence is omitted. We note that the rule indeed can be generalized in different ways, for instance, the condition of $\alpha(\pi(F_1)) = 1$ can be weakened. It is not our focus to formulate a most general rule – instead, a relatively simple, but sufficient, rule is favored.

4.2 Design of Property Verification

The foremost interested property of the protocol is its correctness, i.e., whether or not a unique leader will be elected finally. As discussed previously, we leverage assume-guarantee verification technique, and the assumption is the lower layer model can eventually elect a unique head within a cluster. We will verify this property first in the lower layer, and then verify the upper layer model. Since the two elections are conducted sequentially, this feature simplifies the compositional verification greatly.

Before starting verification, we need to set up a signal indicating the fact that a unique leader of a cluster/system is elected. As we know, if and only if all the variables CH or leader have the same value, then it can be regarded that a cluster head or leader has been elected. Therefore, we label this state as "elected". There are several ways to specify the state "elected". In our experiment, we can calculate the minimum probability of successfully electing a cluster head or leader. If the minimum probability equals to 1.0 (100%), it means that such "elected" is always reachable. Thus the property is expressed as: P_{min} *= ? [F "elected"].* On the other hand, we can also specify the property in other ways, for example, from the initial state, all paths can eventually reach the same destination that the "elected" state is satisfied. In this way, the property can be written as: *P >= 1 [F "elected"].*

In our experiment, we adopted the first approach to calculate the probability of reaching the final "elected" state. To scale up, we set up two parameters, i.e., "N" and "M", where "N" is the number of processes within a cluster, and "M" is the number of clusters of the system. Moreover, we also measured the number of explored states, transitions and the time needed to perform the complete verification. The experiments were conducted on a PC with an Intel i7-4790 processor of 3.6GHz and 32.0GB RAM running Ubuntu 14.04 LTS OS. The versions of PRISM and JDK are 4.3.1 and 1.8.0 respectively. To maximize the performance, in our experiments, the option values of “PRISM_JAVAMAXMEM", “PRISM_JAVASTACKSIZE" and “CUDD" are configured to “Xmx20g", “Xss1536m" and “10g" respectively.

The results of lower layer model verification is given in Table 3. The number of processes is denoted as N. From the table, we can find that across all the configurations with the number of processes N ranging from 6 to 10, a unique head with probability value 1.0 was eventually elected. Our model can scale up to 10 processes given the experiment environment described above with the value of α equal to $\llcorner N/2 \lrcorner + 1$. Time is measured in the unit of seconds, and consists of two parts, i.e., model construction and verification.

The verification result of lower layer protocol indicates that the assumption of successful election in lower layer holds. Then we proceed to verify the upper layer model. The experiment result is given in Table 4. The number of clusters is denoted as M and the time unit is second. From the table, we can observe that as long as the assumption holds, the probability for the system to reach the "elected" state is 1.0. Based on the principle of assume-guarantee technique, we prove that a global leader could be eventually elected by our protocol from the formal verification perspective. Since the maximum number of processes in a cluster and the maximum number of clusters are both 10, our model can support up to 100 processes in total.

Table 3 Verification in lower layer (NC: not counted)

N	Results	State No.	Transition No.	Time(construction , verifiation)(second)
6	1.0	$1.596 * 10^{7}$	$6.054 * 10^{7}$	(5.98, 4.34)
7	1.0	$2.222 * 10^{8}$	$1.012 * 10^{9}$	(20.20, 15.08)
8	1.0	$3.095 * 10^{9}$	$1.658 * 10^{10}$	(151.30, 54.25)
9	1.0	$4.322 * 10^{10}$	$2.675 * 10^{11}$	(1844.60, 286.63)
10	1.0	$6.057 * 10^{11}$	$4.261 * 10^{12}$	(19056.65, 1389.92)
11	NC	NC	NC	NC

Table 4 Verification in upper layer

M	Results	State No.	Transition No.	Time(construction , verifiation)(second)
6	1.0	$1.772 * 10^{6}$	$6.352 * 10^{6}$	(1.96, 2.48)
7	1.0	$1.766 * 10^{7}$	$7.405 * 10^{7}$	(5.42, 2.61)
8	1.0	$1.765 * 10^{8}$	$8.490 * 10^{8}$	(25.14, 7.21)
9	1.0	$1.768 * 10^{9}$	$9.603 * 10^{9}$	(179.21, 26.46)
10	1.0	$1.766 * 10^{10}$	$1.074 * 10^{11}$	(4083.19, 169.87)
11	NC	NC	NC	NC

In order to demonstrate the superior productivity of property verification by separate layers, we conduct a contrast experiment with the verification of a holistic manner. The experiment is based on a system combining with the lower and the upper layer part, and its target is to elect a global leader of the system directly. As the system is influenced by two important factors, the number of clusters and the number of processes of each cluster, we should design the experiment comprehensively considering both factors at the same time. Firstly, we set the cluster number and gradually increase the average number of processes within one cluster. Next we increase the cluster number and repeat the step. The experiment result is summarized in Table 5 in which *CNo.* denotes the number of clusters and *PNo.* denotes the number of processes in one cluster. From Table 5 we can find the number of processes of holistic verification is around 24. By using compositional reasoning, we can increase to 100, and get four times enhancement in terms of scalability.

Table 5 Holistic verification (NC: Not Counted)

CNo., PNo.	Results	State No.	Transition No.	Time(construction , verifiation)(second)
4, 3	1.0	$5.092 * 10^{11}$	$4.668 * 10^{12}$	(1.03, 4.27)
4, 4	1.0	$1.551 * 10^{14}$	$1.875 * 10^{15}$	(2.50, 10.99)
4, 5	NC	NC	NC	NC
5, 4	1.0	$1.844 * 10^{19}$	$2.825 * 10^{20}$	(6.56, 32.00)
5, 5	NC	NC	NC	NC
6, 4	1.0	$1.162 * 10^{23}$	$2.136 * 10^{24}$	(15.33, 66.28)
6, 5	NC	NC	NC	NC

4.3 Extension of rewards

To analyze the quantitative properties of the protocol, we extended the model with the energy consumption via cost/rewards, and conducted the second experiment. Energy consumption is an important concern, especially for wireless sensor networks [18, 19]. Thus the energy consumption analysis model of a protocol is much needed. Based on the work [20] and many others [21], the energy consumption of internal operation is usually 1.0-1.5 times of communication. Therefore, we assume that the unit energy consumption of communication is 1 and that of the internal operation is 1.5.

In PRISM, by the concepts of costs and rewards, we augment the transition steps with real values denoting the energy consumption. Listing 12 describes the part of the augmented model. The communication happens when processes exchange the messages, for example, TRUST messages and *rec_from* messages, so we add the rewards statements with the synchronization label.

Listing 12 Modeling the energy consumption with Rewards

```
rewards "consumption"
//execute TRUST operations
[A1] true : 1; ...
[An] true : 1;
//update leader info. of every cluster
[A1_1] true : 1; ...
[An_1] true: 1;
//internal operation rewards
[] true : 1.5; ...
endrewards
```

After assigning the rewards to the transitions, then we can calculate the accumulated energy consumption of selecting an eventual leader. In PRISM, we specify as a property, i.e., R_{min} = *? [F "elected"]* and R_{max} = *? [F "elected"]*, where R_{min} denotes the minimum accumulated rewards to reach the specified states and R_{max} denotes the maximum rewards. The experiment is conducted in the lower layer and upper layer separately, since both layers contain the election steps. Table 6 and 7 summarize the energy consumptions in lower layer and upper layer respectively in which the minimum rewards, the maximum rewards and the average rewards of electing a leader are given. From the tables, we could also observe that the scale of models suffers because of the additional states and calculations introduced by rewards.

Table 6 Energy consumption in lower layer (NC: Not Counted)

Process No.	Minimum	Maximum	Average
5	15.622	58.916	37.269
6	17.496	90.988	54.242
7	NC	NC	NC

4.4 Extension of unreliable channel

As aforementioned, the original model holds a strong assumption that the underlying network is reliable. But real networks are subject to accidental events and message lost during transmission happens frequently. We extend the protocol and modeling the channel reliability with probability. Particularly, we use a single probability variable to present the channels of both layers, and use quantitative analysis to calculate the numerical possibility of a successful eventual election. Then we can measure the relationship between the channel reliability and the election result quantitatively. If the probability of successful election of lower layer is p_1 and that of upper layer is p_2, based on the product principle, we can get the final probability of global election by p_1*p_2.

Table 7 Energy consumption in upper layer (NC: Not Counted)

Cluster Head No.	Minimum	Maximum	Average
5	33.518	38.469	35.993
6	39.078	44.338	41.708
7	44.156	50.036	47.096
8	NC	NC	NC

Listing 13 Unreliable communication channel modeling

```
[s] (time=true)&(success=false)&(s1=0) ->
    p2:(s1'=1)+p3:(s1'=1)&(R1_1'=false)+
    p1:(s1'=0)&(time1'=time1+1);
...
formula time=(time1<=t)&(time2<=t)...(timen<=t);
```

Listing 13 illustrates our modeling for unreliable communication channels partially. For example, in the QUERY message transmission phase, it is possible that the message gets lost. We assign it a value p_1 to represent the probability of message loss. Moreover, we define a threshold t for the number of re-sending the message and set an integer variable *timeN* for each process (N shall be replaced with the process id) to count the number of fault times. If the value of t is zero, then it reduces to the case of best-effort service where there is no re-sending mechanism applied. Otherwise, when the value of $time1$ is equal to that of t, it represents a fault and the process will not retry.

In this experiment, we fix the number of clusters and processes but make the reliability and re-sending times as variables. Particularly, in each cluster, we have four processes, and the α value is three. The number of clusters are four. The experiment consists of two parts. In the first part, we set the retry number to be zero, i.e., re-sending the message is not allowed in case of message loss. The message loss probability changes from 0.005 to 0.2, and the resulting election success probability drops from 0.8778 to 0.0719 accordingly. Table 8 summarizes the relation between the message loss rate and the election success probability. Figure 1 illustrates this pictorially.

In the second part of the experiment, we add the factor of retry in case of message loss and make the channel of reliability fixed. Particularly, we set the retry value from one to five, and the ratio of message loss to be 0.2. The results are summarized in Table 9 and illustrated in Figure 2. From

Table 8 Election with unreliable channels without retry mechanism

p_1	0.005	0.01	0.05	0.1	0.2
lower layer	0.9801	0.9606	0.8145	0.6561	0.4096
upper layer	0.8956	0.8106	0.4674	0.3130	0.1756
total	0.8778	0.7787	0.3807	0.2054	0.0719

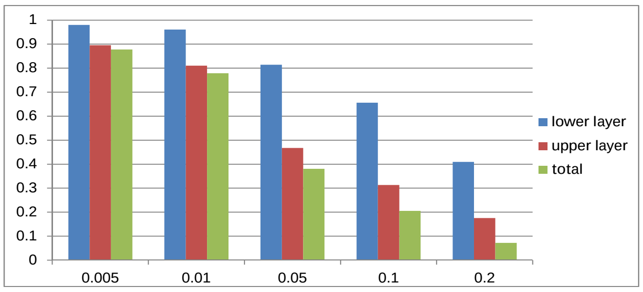

Fig. 1 Experimental results without retry mechanism

the table, we can observe that the value of successful election probability increases gradually as the retry times go up.

Table 9 Election with unreliable channels with retry mechanism

retry times	1	2	3	4	5
lower layer	0.8493	0.9684	0.9961	0.9994	0.9999
upper layer	0.5310	0.7696	0.9208	0.9796	0.9944
total	0.4510	0.7453	0.9172	0.9790	0.9943

5 Discussion

As mentioned before, PRISM model checker uses a state-transition based language which is quite different from imperative languages. Therefore, we have to abstract away some operation details during the modeling phase. Secondly, the protocol contains many set operations, such as union and intersection. The set data structure is not supported by the PRISM modeling language explicitly. We use Boolean bit vectors to mimic the set operation. Concretely, we associate each set element with a corresponding Boolean Vector. Since we know the value of the elements in the set beforehand, if it appears in the set, we set the corresponding Boolean bit to be true, and otherwise false. For example, we have two Sets, i.e., $A = \{1,2,3\}$, $B = \{3,4\}$, then we define a superset as $S = \{1,2,3,4\}$ and a Boolean bit vector for each set accordingly. Thus for set A, the bit vector is $\{true, true, true, false\}$, and for set B, the bit vector is $\{false, false, true, true\}$. Therefore, to calculate $A \cup B$, we can set the bit vector to be $\{true, true, true, true\}$; to calculate $A \cap B$, we set the bit vector to be $\{false, false, true, false\}$.

Listing 14 Communication pattern illustration

```
[A2] (s2=3) & (TRUST2=true) -> action ...;
[A2] (s2=3) & (TRUST2=false) -> action ...;
```

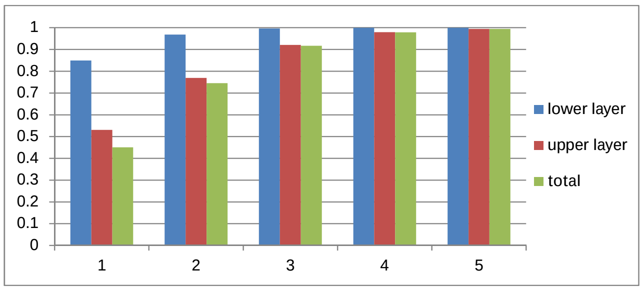

Fig. 2 Experimental results with retry mechanism

Moreover, in our verification experiment, we need to measure the performance of systems with different number of processes. However, PRISM does not provide the template functionality to facilitate scaling-up the model. Manual coding is time-consuming and error-prone. We observe that in the model, the processes are essentially the same except for their identities and the operations are quite similar. Hence, we leverage a code generation technique to automate this process. Concretely, we extract the patterns of internal operation and external communication, and leverage the rule-based code generation technique to output the corresponding PRISM models. For example, given a cluster of 10 processes, a process needs to monitor the QUERY or TRUST messages from other processes, the code pattern is illustrated in Listing 14. Although the number of statements can be different among different system, it is associated with the number of processes. Thus we can make the process number as the parameter and automate the system model generation process. The parameterized model generation as well as the above abstraction technique can also be used in other similar protocol verification settings.

6 Related Work

There are two relevant threads of research in the field. One is about the leader election protocol design in distributed computing, and the other is the formal analysis of protocols based on model checking. In this section, we briefly review some representative work in each category.

In [6], Mostefaoui et al. adapted an existing leader election protocol designed for static systems and then customize it so as to work in dynamic systems. The correctness of the protocol is established through theoretical proof. Different from our work, the protocol cannot be applied in the hierarchical cluster based settings and neither did they give a quantitative analysis of their work based on formal verification. Larrea et al. [7] considered the eventual leader election in an asynchronous system prone to process crashes and they proposed a specification of Ω suited to dynamic systems. Then they verified its correctness and introduced the notation of an eventual leader suited to dynamic systems and simultaneously an additional property related to the stability of systems.

Gupta et al. [22] proposed a scalable leader election protocol suitable for large process groups under a weak membership requirement. The protocol supported quite good guarantees about termination in the sense of the classical specification of the election problem and of generating a fixed number of processes, both independent of group size. Different from our work, neither of the above two approaches considered quantitative properties based on formal verification. In [23], Tang et al. proposed a novel leader election protocol to cope with the situations where a unique identify for a process is not always possible. But it is constrained within the scope of static systems instead of dynamic systems as considered in our work.

Meanwhile, probabilistic model checking, as a typical method of formal verification, plays an important role in the verification against the safety and security properties of various kinds of protocols. In [24], Duflot et al. provided an overview of this area and discussed two approaches to the implementation of quantitative verification of these protocols based on probabilistic model checking. Baier et al. [25] focused on the strength and limitations of probabilistic model checking in the context of a multi-disciplinary project, in which they applied formal approaches for reasoning about energy-awareness and other quantitative aspects of low-level resource management protocols. Naskos et al. [26] concentrated on the on-demand resource provisioning in cloud computing, referred as cloud elasticity. They proposed a method about the development of more formalized and dependable elasticity policies and presented an extensible way to enforce elasticity through the dynamic instantiation and online quantitative verification of Markov decision process by using probabilistic model checking. In [27], He et al. leveraged the probabilistic model checking to verify a newly proposed pipe protocol in the domain of Internet-of-Things where quantitative analysis can be conducted. The application of probabilistic verification of these works is similar to ours, but they did not deal with the consensus problems addressed by eventual leader election protocols in distributed computing. Zhang et al. [28] proposed a method to verify properties of the Timing-sync Protocol for Sensor Netowrks (TPSN). Different from ours, the quantitative aspect of verification is mainly conducted based on statistical verification, which is essentially a simulation-based approach. The quantitative analysis of eventual leader election protocols by probabilistic model checking were conducted in [5, 13, 20]. But in all these works, either they only dealt with the static environment settings [5, 20], or they did not consider the hierarchical cluster based system models [13].

7 Conclusion and Future Work

Recently, a lot of efforts have been given to the design of eventual leader election protocols in different environmental settings, but relatively little emphasis has been given to the formal verification and quantitative analysis. In this paper, we complement this by using probabilistic model checking to verify a newly proposed hierarchical leader election protocol for dynamic system. Particularly, we use a compositional verification technique, i.e., assume-guarantee to verify the two layers respectively, and demonstrate a performance boost compared with holistic verification. We also augment the original model with additional features by cost/rewards, and with unreliable communication channels by probability, and thus more quantitative properties can be analyzed, such as energy consumption, and the relation between eventual election and the message loss rate. Although we only consider the hierarchical leader election protocol in our paper, the technique can also be extended to model and verify other similar consensus protocols or resource-restricted routing protocols [14, 18] in wireless sensor networks.

To the best of our knowledge, this is the first work to apply quantitative verification techniques to the hierarchical leader election protocol design. But the potential of probabilistic model checking has not been fully exploited, and future research possibilities include more detailed investigation on the fully dynamic settings prior to arriving at the stable phase, and the property analysis on the model of more sophisticated reliability control mechanisms against communication channels.

Acknowledgements This work was partially supported by the Fundamental Research Funds for the Central Universities under grant No. NS2016093.

References

1. Sara Tucci-Piergiovanni and Roberto Baldoni. Eventual leader election in infinite arrival message-passing system model with bounded concurrency. In *Dependable Computing Conference (EDCC), 2010 European*, pages 127–134. IEEE, 2010.
2. Gurdip Singh. Leader election in the presence of link failures. *IEEE Transactions on Parallel & Distributed Systems*, (3):231–236, 1996.
3. Koji Nakano and Stephan Olariu. A survey on leader election protocols for radio networks. In *Parallel Architectures, Algorithms and Networks, 2002. I-SPAN'02. Proceedings. International Symposium on*, pages 63–68. IEEE, 2002.
4. Michael Fischer and Hong Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In *Principles of Distributed Systems*, pages 395–409. Springer, 2006.
5. Rena Bakhshi, Wan Fokkink, Jun Pang, and Jaco Van De Pol. Leader election in anonymous rings: Franklin goes probabilistic. In *Fifth Ifip International Conference On Theoretical Computer Science–Tcs 2008*, pages 57–72. Springer, 2008.
6. Achour Mostefaoui, Michel Raynal, Corentin Travers, Stacy Patterson, Divyakant Agrawal, and AE Abbadi. From static distributed systems to dynamic systems. In *Reliable Distributed Systems, 2005. SRDS 2005. 24th IEEE Symposium on*, pages 109–118. IEEE, 2005.
7. Mikel Larrea, Michel Raynal, Iratxe Soraluze, and Roberto Cortiñas. Specifying and implementing an eventual leader service for dynamic

systems. *International Journal of Web and Grid Services*, 8(3):204–224, 2012.

8. Carlos Gómez-Calzado, Alberto Lafuente, Mikel Larrea, and Michel Raynal. Fault-tolerant leader election in mobile dynamic distributed systems. In *Dependable Computing (PRDC), 2013 IEEE 19th Pacific Rim International Symposium on*, pages 78–87. IEEE, 2013.
9. Huaguan Li, Weigang Wu, and Yu Zhou. Hierarchical eventual leader election for dynamic systems. In *Algorithms and Architectures for Parallel Processing*, pages 338–351. LNCS, vol.8630, Springer, 2014.
10. Christel Baier, Joost-Pieter Katoen, et al. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.
11. Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, and David Parker. Automated verification techniques for probabilistic systems. In *Formal Methods for Eternal Networked Software Systems*, pages 53–113. Springer, 2011.
12. Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: verification of probabilistic real-time systems. *Lecture Notes in Computer Science*, 6806:585–591, 2011.
13. Jiayi Gu, Yu Zhou, Taolue Chen, and Weigang Wu. Analyzing eventual leader election protocols for dynamic systems by probabilistic model checking. In *Cloud Computing and Security*, pages 192–205. LNCS, vol.9483, Springer, 2015.
14. Weigang Wu, Jiannong Cao, and Michel Raynal. Eventual clusterer: A modular approach to designing hierarchical consensus protocols in manets. *Parallel and Distributed Systems, IEEE Transactions on*, 20(6):753–765, 2009.
15. Zhiwei Yang, Weigang Wu, Yishun Chen, and Jun Zhang. Efficient information dissemination in dynamic networks. In *Parallel Processing (ICPP), 2013 42nd International Conference on*, pages 603–610. IEEE, 2013.
16. Marta Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. Assume-guarantee verification for probabilistic systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 23–37. Springer, 2010.
17. Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. Compositional probabilistic verification through multi-objective model checking. *Inf. Comput.*, 232:38–65, 2013.
18. Jian Shen, Haowen Tan, Jin Wang, Jinwei Wang, and Sungyoung Lee. A novel routing protocol providing good transmission reliability in underwater sensor networks. *Journal of Internet Technology*, 16(1):171–178, 2015.
19. Shengdong Xie and Yuxiang Wang. Construction of tree network with limited delivery latency in homogeneous wireless sensor networks. *Wireless personal communications*, 78(1):231–246, 2014.
20. Haidi Yue and Joost-Pieter Katoen. Leader election in anonymous radio networks: model checking energy consumption. In *Analytical and Stochastic Modeling Techniques and Applications*, pages 247–261. Springer, 2010.
21. Tifenn Rault, Abdelmadjid Bouabdallah, and Yacine Challal. Energy efficiency in wireless sensor networks: A top-down survey. *Computer Networks*, 67:104–122, 2014.
22. Indranil Gupta, Robbert Van Renesse, and Kenneth P Birman. A probabilistically correct leader election protocol for large groups. In *Distributed Computing*, pages 89–103. Springer, 2000.
23. Jian Tang, Ernesto Jiménez, Carlos Herrera, and Sergio Arévalo Viñuales. Eventual election of multiple leaders for solving consensus in anonymous systems. *The Journal of Supercomputing*, (10):1–18, 2015.
24. Marie Duflot, Marta Kwiatkowska, Gethin Norman, David Parker, Sylvain Peyronnet, Claudine Picaronny, and Jeremy Sproston. Practical applications of probabilistic model checking to communication protocols. 2012.
25. Christel Baier, Clemens Dubslaff, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. Probabilistic model checking for energy-utility analysis. In *Horizons of the Mind. A Tribute to Prakash Panangaden*, pages 96–123. Springer, 2014.
26. Athanasios Naskos, Emmanouela Stachtiari, Anastasios Gounaris, Panagiotis Katsaros, Dimitrios Tsoumakos, Ioannis Konstantinou, and Spyros Sioutas. Dependable horizontal scaling based on probabilistic model checking. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 31–40. IEEE, 2015.
27. Kangli He, Min Zhang, Jia He, and Yixiang Chen. Probabilistic model checking of pipe protocol. In *Theoretical Aspects of Software Engineering (TASE), 2015 International Symposium on*, pages 135–138. IEEE, 2015.
28. Fengling Zhang, Lei Bu, Linzhang Wang, Jianhua Zhao, Xin Chen, Tian Zhang, and Xuandong Li. Modeling and evaluation of wireless sensor network protocols by stochastic timed automata. *Electronic Notes in Theoretical Computer Science*, 296:261–277, 2013.