Parallel Passive Testing of System Protocols - Towards a Real-time Exhaustive Approach

Baptiste Alcalde and Ana Cavalli
Institut National des Télécommunications GET-INT
Evry, France
{baptiste.alcalde, ana.cavalli}@int-evry.fr

Abstract

Passive testing has proved to be a powerful technique for protocol system fault detection by observing its input/output behaviors yet without interrupting its normal operations. Various techniques for passive testing on Extended Finite State Machine (EFSM) exist such as by invariants and by interval determination, which are not very costy in terms of complexity but that also don't detect every errors. To improve the fault detection capabilities a backward checking method was proposed. It analyzes in a backward fashion the input/output trace from passive testing and its past. It effectively checks both the control and data portion of a protocol system and is able to detect every errors, but with a higher complexity. The purpose of the present paper is then to propose possible parallel algorithms and architectures to improve the backward passive testing approach by narrowing down its maximal time complexity and transforming it into the first real-time exhaustive passive testing approach. We present the backward algorithm and its parallel versions, study their complexity, and report results on various communication and routing protocols.

1. Introduction

Passive testing is an activity of detecting faults in a system under test by observing its input/output behaviors without interfering its normal operations. The usual approach of passive testing consists of recording the trace produced by the implementation under test and trying to find a fault by comparing this trace with the specification ([7], [9], [13]). Other approaches explore relevant properties required for a correct implementation, which are named invariants, and then check them on the implementation traces of the systems under test ([2], [3], [14]). Most of the works on passive testing are based on finite state machines (FSMs) and are focused on the control part of the tested systems without taking into account data parts. To cope with protocol data portions, Extended Finite State Machines (EFSMs) are used to model the systems, which include parameters and variables to encode data. In [13] a first approach to perform passive testing on EFSMs was proposed. An algorithm based on constraints on variables was developed and applied to GSM-MAP protocol. However, this algorithm cannot detect transfer errors. In [6], an algorithm based on variable determination with the constraints on variables was presented. This algorithm allows to trace the variables values as well as the system state, however, every transfer errors still cannot be detected. To overcome this limitation, a new approach based on backward tracing was proposed in [1]. This algorithm processes the trace in an antichronological order to further narrow down the possible configurations for the beginning of the trace and to continue the exploration in the past of the trace with the help of the specification. This algorithm contains two phases. First, it follows a given trace backward to find the possible initial configurations at the beginning of the trace. Secondly, it starts from these configurations and explore backward every possible path of the specification, with help of prunning operations and a transition choice strategy.

As written in [1], the maximal complexity can be considered as high even if it is in fact low comparatively with other exhaustive techniques. To reduce this time complexity, and increase the power of the passive testing approach, we propose to parallelize this algorithm. A parallel algorithm, as opposed to a traditional serial algorithm, is one which can be executed a piece at a time in many different processing devices, and then put back together again at the end to get the correct result. Parallel algorithms are valuable because it is far easier to execute large computing tasks via a parallel algorithm than it is via a serial (non-parallel) algorithm, given the way all modern processors work. Technically, it is far more difficult to construct a fast single processor than have many but slow processors with the same throughput. There are also certain theoretical limits to the potential of serial processors [8].

The rest of the paper is organized as follows. Section 2 describes the basic concepts used in the paper. Section 3 reminds the concepts of the backward checking algorithm. In section 4 the parallelized versions are proposed. Complexity of the algorithms is discussed in section 5, and section 6 reports the results of their application to the communication protocols SCP, INRES, and the routing protocols OLSR, and OSPF to show their efficiency and the consequences on the required test architecture.

2. Preliminaries

We first introduce basic concepts needed.

2.1. Extended Finite State Machine

We use Extended Finite State Machine (EFSM) to model the system protocols.

An Extended Finite State Machine M is a 6-tuple $M = < S, s_0, I, O, \vec{x}, T >$ where S is a finite set of states, s_0 is the initial state, I is a finite set of input symbols (eventually with parameters), O is a finite set of output symbols (eventually with parameters), $\vec{x}$ is a vector denoting a finite set of variables, and T is a finite set of transitions.

A transition t is a 6-tuple $t =< s_i, s_f, i, o, P, A >$ where s_i and s_f are the initial and final states of the transition respectively, i and o are the input and the output, P is the predicate (a boolean expression), and A is the ordered set (sequence) of actions.

In the present work we consider that a variable is defined in a finite interval of N, a parameter is a value in N which will be mapped to a variable, and an action is an affectation of the variable v of the form : $v = \sum_1^n a_i x_i$, where $a_i \in N$, $x_i \in \vec{x}$, and n is the number of variables in the set $\vec{x}$.

The inputs and outputs are the (eventually parametrized) symbols respectively received or produced by the observed system. The predicate denotes the condition under which the transition can be performed.

An event trace is a sequence of I/O pairs. In this paper we consider that the traces can start at any moment of the implementation execution.

Given a trace from the implementation under test and a specification and according to the passive testing litterature, we can face three types of errors, which are defined as follow :

1. **output errors** : when the output of a transition in the implementation differs from the output of the corresponding transition in the specification.

2. **transfer errors** : when the ending state of a transition in the implementation differs from the ending state of the corresponding transition in the specification.

3. **mixed errors** : when the output and the ending state of a transition in the implementation differ from the output and the ending state of the corresponding transition in the specification.

The purpose of this paper is limited to the error detection. The fault identification and fault localisation topics are not taken into account in this work. Thus, when an error will be detected there is no information about its type.

2.2. Candidate Configuration Set

For the passive testing approach we need a structure to keep information about contraints and location of the studied traces, which is named Candidate Configuration Set, according to [6].

Let M be an EFSM. A Candidate Configuration Set (CCS), is a 3-tuple $(e, R, Assert)$ where e is a state of M, R is an environment, and $Assert$ is an assertion, that is to say a boolean expression on variables. R records the current interval of each variable of $\vec{x}$. The assertion are used to record additional information, i.e., bindings with other variables or excluded values.

In the backward checking approach we interest ourselves in confirming a set of variables and then we extend the concept of CCS by the following definitions.

A variable v is validated (also called confirmed) when we find in the processed transition the information saying it must be in a set of value I and we have the relation : $I \subset I_c$, where I_c is the current set of values of v.

A variable is called determinant in the past of the trace if it was not validated yet.

Let M be an EFSM. An Extended Candidate Configuration Set (ECCS) is a 4-tuple $(e, R, Assert, D)$, where e is a state of M, R is an environment, $Assert$ is an assertion, and D is a set of determinant variables.

A configuration is validated (also called confirmed) when it contains no more determinant (the set D is empty).

3. The Backward Checking Approach

3.1. Overview

The Backward Checking algorithm - as it was presented in [1] is an approach of passive testing on EFSMs derived from the testing by determination of the intervals of variables such as the precursor one of [6]. In this kind of passive testing, the variables are defined in intervals and the goal is to find inconsistency between these intervals and the information given in the trace exploration about these variables. We consider that we have a system under test on which we place an observation point. We suppose that this observation point records the event traces respecting their causal

order. We assume that finding the order of these events is a well studied and resolved problem.

We also consider that the observation can start at any moment, without any preliminary operations. In particular we don't place the system under test into a known configuration because it would mean that we somehow control the system. If we control or interact with the system then we enter in an active testing architecture - the topic of which is beyond the scope of the present paper - and the problem can be solved according to the works of this community.

The Backward Checking algorithm is composed of two main phases. First, it follows a given event trace backward to find the possible initial configurations at the beginning of the trace. Secondly, it starts from these configurations and explore backward every possible path of the specification, with help of prunning operations and a transition choice strategy, to reduce as much as possible the search.

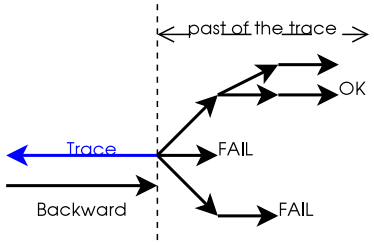

Figure 1. Overview of Backward Checking

3.2. In the trace

During the first phase, if an inconsistency is detected it means that the event trace is not correct, and then an error had been detected. If, in the end of the event trace analysis, we didn't find an inconsistency, it means the studied trace is possible with the obtained initial configurations. Then, we launch the second phase, that is to say the exploration of the past of this trace.

3.3. In the past of the trace

In the second phase we try to confirm the intervals in which the variables are defined according to the initial configurations. The algorithm finishes with a positive answer (trace is valid) at the first confirmed configuration, that is to say the first configuration in which every variables had been confirmed, or with a negative answer (invalid trace) if every branch of the exploration tree leads to an inconsistency. We can say that this algorithm is optimistic in the way that it will be fast to say that there is no error, but slower to say that there is one. This fact is coherent with reality because we suppose that an error in an event trace is an exceptional behaviour.

The different branches of the exploration tree are the possible successions of transitions, taken in a backward manner, from the initial configurations resulting from the first phase.

To avoid loops and reduce the size of the new configurations we use the *privation* operator. It is defined in [1] as follows :

Given four ECCS $c_1 = (e, R_1, Assert_1, D)$, $c_2 = (e, R_2, Assert_2, D)$, $c_a = (e, R_a, Assert_a, D)$ and $c_b = (e, R_b, Assert_b, D)$. Doing $c_1 \backslash c_2$ produces c_a and c_b such that :

1. for c_a :
   (a) for each variable v, we have got : $R_a(v) = R_1(v) \cap R_2(v)$ where $\cap$ is the intervals intersection operator,
   (b) $Assert_a = Assert_1 \wedge \overline{Assert_2}$, where $\wedge$ is the boolean "and" operator,
2. for c_b :
   (a) $R_b = R_1$,
   (b) $Assert_b = Assert_1 \wedge (\bigvee_{i=0}^{|V|-1} (v_i \notin R_2(v_i)))$ where $\wedge$ is the boolean "and" operator, and $\vee$ is the boolean "or" operator (be careful of priorities of parenthesis).

Let E_1 and E_2 be the cartesian products of the intervals of R_1 and R_2 respectively. The aim of the privation operator is to obtain the $E_1 \setminus E_2$ set - where $\setminus$ is the set minus symbol - possibly taking into account the assertion constraints. Therefore the ECCS c_a and c_b will contain the cases of c_1 that were not yet processed in c_2. c_a deals with the intersection of c_1 and c_2 whereas c_b is the rest of c_1 (cf Fig.2).

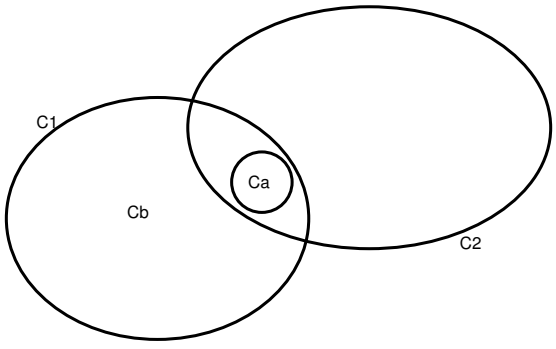

Figure 2.

3.4. Transition Choice Strategy

In addition the transitions are choosen through a process of selection depending on several criteria. This strategy is the following: the transitions are classified by order or priority where the biggest priority means the most chances this transition can lead to a verdict. The three criteria are :

- the starting state of the transition is the initial state of the EFSM. It is obvious that if we arrive at the initial state of the specification, the variables will be confirmed or leading to an inconsistency;
- the starting state of the transition is a state that was never seen in a former configuration. In this case we open more paths if the transition is successfully processed;
- the transition "determines" a variable, that is to say there is an action involving the variable v and v is the left member of the action. It is clearly an attempt to confirm the variable v.

For each criterion, a weight is given (empirically). Lets consider that the weights are named respectivelely w_1, w_2 and w_3 for the three criteria seen above and a is the number of actions in the transition. The global priority W_t is then given by the following formula :

$$W_t = w_1 + w_2 + a.w_3$$

3.5. The main algorithm

Considering all the concepts defined above we can present in the following the main algorithms. The algorithm for the trace simply consists in backtracking transitions along the trace, and in invaliding it if an inconsistency is found. The algorithm for the past of the trace is also a transition backtracking but without guidance of events (no more trace), associated with the operations to reduce search space and the Transition Choice Strategy.

4. Parallelized Backward Checking

In this section we discuss the parallel version of Backward Checking algorithm and its benefits on the complexity.

In order to improve furthermore the efficiency of the backward checking algorithm we present the following algorithms. As we can deduct from the study of the backward checking algorithm, the first part (the trace) will be treated in parallel but without reducing the complexity given the fact that it is closely bound to the trace. Actually, we saw it is proportional to the length of the trace, so we can only reduce the proportion multiplicator with the parallel version. The most interesting problematic is to parallelize the second part of the algorithm (past of the trace). In the previous sections we presented the process of the algorithm in the past and we presented the operations on the configurations. We have to make an important remark at this point of the paper : the operations provide us a way to get independent configurations. In fact, if two configurations are dependent on one another it means that a total or partial inclusion exists between them - this will be detected by intersection and privation operations. The study of configurations is then possible in parallel. First, we present three variants of the parallel approach, and then we discuss their advantages and drawbacks.

For each of the following algorithms we have a parallel loop that computes the predecessor set of an ECCS. It means that if we dispose of a set of ECCS $(c_1, \ldots, c_n)$, all these configurations are processed in parallel instead of being processed sequencially. Nevertheless, we must check out and erase eventual redundancies after obtaining the predecessor set.

We can distinguish two methods to treat the parallelism: the first is a parallel algorithm by round and the second is a parallel algorithm without round that we call here direct algorithm. The method by round treats a set of transitions in parallel with a system of synchronization after each round whereas the direct method doesn't use synchronization.

For the redundancy checking in the method by round we consider two main ways to do. Either we check each new configuration with all the configurations seen till the last round, and then check the configurations of the current round between themselves, or we do the same in the opposite order.

So, finally, there are three main ways to parallelize the process : two by round and one direct. Now, we present these algorithms.

For the following algorithms we consider that Q is the set of configuration-transition pairs to be explored, V is the set of already-seen Extended Configurations, and X is the set of Extended Configurations seen in the current round. Also $Back\_past\_transition(t, c)$ is a function that takes a configuration c and backtracks transition t returning the new configuration c', $Check\_redundancy(c', V)$ takes a configuration c' and checks its inclusions with each configurations of the set V, and $Check\_redundancy\_till(c', X, i)$ compares c' only with the i first configurations of the set X (i excluded). More details about these operations can be found in [1]. Also, $c'.D$ denotes the determinant set D of the configuration c'. The algorithms return TRUE if the trace is correct, FALSE if it is not.

The first presented algorithm is the one proposed for a processing with rounds and checking first the current round found configurations with the configurations seen till the last round, the second one is by round and checking first

the current round found configurations between themselves. The third presented algorithm is the direct one.

4.1. The first method by round

```
initialize Q, V, X
while Q ≠ ∅ do
    parallely for each i-processor do
        take the i-th item < c, t > from Q
        c' ⟵Back_past_transition(t, c)
        if c' ≠ ∅ do
            if c'.D = ∅ do
                return TRUE
            c' =Check_redundancy(c', V)
            if c' ≠ ∅ do
                X ⟵ X ∪ c'
    parallely for each i-processor do
        c' = X_i
        c' =Check_redundancy_till(c', X, i)
        if c' ≠ ∅ do
            V ⟵ V ∪ c'
            parallely for each transition t
              where t.end_state = c'.state do
                calculate the weight of < c', t >
                insert < c', t > into Q by its weight
return FALSE
```

4.2. The second method by round

```
initialize Q, V, X
while Q ≠ ∅ do
    parallely for each i-processor do
        take the i-th item < c, t > from Q
        c' ⟵Back_past_transition(t, c)
        if c' ≠ ∅ do
            if c'.D = ∅ do
                return TRUE
            X ⟵ X ∪ c'
    parallely for each i-processor do
        c' = X_i  c' =Check_redundancy_till(c', X, i)
        if c' ≠ ∅ do
            c' =Check_redundancy(c', V)
            if c' ≠ ∅ do
                V ⟵ V ∪ c'
                parallely for each transition t
                  where t.end_state = c'.state do
                    calculate the weight of < c', t >
                    insert < c', t > into Q by its weight
return FALSE
```

4.3. Direct parallel method

```
initialize Q, V
while Q ≠ ∅ do
    parallely for each processor do
        take the first item < c, t > from Q
        c' ⟵Back_past_transition(t, c)
        if c' ≠ ∅ do
            if c'.D = ∅ do
                return TRUE
            sequencially do
                c' =Check_redundancy(c', V)
                if c' ≠ ∅ do
                    V ⟵ V ∪ c'
            if c' ≠ ∅ do
                parallely for each transition t
                  where t.end_state = c'.state do
                    calculate the weight of < c', t >
                    insert < c', t > into Q by its weight
return FALSE
```

5. Complexity

In the present section we discuss about the advantages of the different algorithms and about the complexity of both normal and parallel algorithms.

5.1. Compared study of the proposed algorithms

In the previous subsection we proposed three algorithms. Each one has advantages and drawbacks that the present subsection wants to expose in order to define which method looks the most interesting according to the specification.

First, we can emit a few comments about the number of needed processors for the two methods (round and direct).

For a round method, the number of processors can be limited to the number of transitions of the EFSM. Indeed, at a given round we can't have more transitions to backtrace than the number of transitions contained in the EFSM. So it's not usefull to use more than $|T|$ processors (where $|T|$ is the number of transitions of the EFSM). Neverthless, for a correct processing the architecture has to prevent a transition from not having a corresponding processor. Finally, we say the method by round needs exactly $|T|$ processors.

For the direct method, as there is no synchronization, we can only limit the number of processors to the number of ECCS the EFSM can produce. This number is big and such a requested architecture is surely not available in today's world for complex protocols or services. However, we don't need so many processors and in the limit case where we have access to only one processor, the process is equivalent

to the former sequencial process (without any gain in performance). So to have an effective gain we need at least two processors, and in order to have an optimal gain we propose to use an architecture with at least $max|T\_in_i|$, the maximal number of entering transitions of a state from the EFSM specification - this to cope with the last parallel loop of the algorithm.

On the other hand, it is interesting to compare the three algorithms in order to show which one is the most efficient in a given situation. Let n be the number of already seen configurations till the i-th round (for the methods by round) or at a given time (for the direct method). We know that in this case the maximal number of transitions to be analysed is equal to the number of transitions contained in the EFSM. Let T be this number. The question we want to answer is which technique will take the less time to analyse t transitions (where $t \in [0;T]$).

For the direct method, we process one transition per processor, then we eventually add new non-empty ECCS in the already seen ECCS set and add the next transitions to be backtracked according to their priority weight. In other words, the already seen configuration set increases incrementally after each processed transition that gives a new non-empty ECCS. Then, the maximal time needed to process the t transitions can be given by :

$$\sum_{k=0}^{t-1}(n+k) = tn + \sum_{k=0}^{t-1} k (= tn + \frac{t(t-1)}{2})$$

For the first method by round (with comparison of ECCS of the current round between themselves first), the maximal number of comparison is :

$$\sum_{k=0}^{t-1} k + xn (= xn + \frac{t(t-1)}{2}),$$

where x is the number of refined ECCS remaining after comparison with $x \in [1;t]$

For the second method by round the formula is :

$$tn + \sum_{k=0}^{y-1} k (= tn + \frac{y(y-1)}{2}),$$

where y is the number of refined ECCS remaining after comparison with $y \in [1;t]$

On the base of the above statements, we can compare the different methods. First, we compare the direct method with the first method by round. The direct method is more efficient for t transitions iff :

$$\begin{array}{c} tn + \sum_{k=0}^{t-1} k \leq \sum_{k=0}^{t-1} k + xn \\ tn \leq xn \end{array}$$

From the above lines we conclude that the direct method is never better than the round one.

In the same way we determine that the direct method is more efficient than the second method by round iff :

$$\begin{array}{c} tn + \sum_{k=0}^{t-1} k \leq tn + \sum_{k=0}^{y-1} k \\ \sum_{k=0}^{t-1} k \leq \sum_{k=0}^{y-1} k \\ t \leq y \end{array}$$

From these results we deduct that the direct method is always slower than any method by round.

We must also compare the two techniques by round and we obtain :

$$\begin{array}{c} \sum_{k=0}^{t-1} k + xn \leq tn + \sum_{k=0}^{y-1} k \\ \sum_{k=y-1}^{t-1} k \leq (t-x)n \\ \frac{t(t-1)-y(y-1)}{2} \leq (t-x)n \end{array}$$

This relation shows us that the first technique is the fastest since there is at least one refinement ($x < t$) and n is big. The number n will increase as the algorithm processes, and its upper bound can be high. Then the first technique seems to be more promising. But if $x = t$ (no refinement) then the first technique is better only if :

$$\sum_{k=y-1}^{t-1} k (= \frac{t(t-1)-y(y-1)}{2}) = 0,$$

that is to say $y = t$ (no refinement in the second technique also). Unfortunately, there is no way to predict the number of refinement in first nor second technique, and we count on experiments to show which one is statistically more interesting.

5.2. Some remarks

First we must note that the techniques by round don't really benefit from the Transition Choice Strategy as every transition of a round is backtracked at the same time. It can then make a sense if the processors are different and classified from the most to the less performant. Then the first transition of the list - the most promissing one - is processed by the most efficient processor and in case the transition brings a positive conclusion ($TRUE$) the parallel algorithm stops immediately.

The direct method suffers from its sequencial part. On the other hand we can remark that this method uses optimally the Transition Choice Strategy. In addition, it can be efficient in case there are lots of unconsistent transitions since the sequencial part is then not processed.

5.3. Complexity of the sequential Backward Checking

In the first part of the algorithm (trace) the complexity depends on the trace. We have according to [1]:

Proposition 5.1 *Suppose that the observed event trace is of length l, then the complexity of the first part of the presented algorithm is proportional to l.*

For the second part (past of the trace) the complexity depends on the number of possible configurations. A configuration includes a state number, interval of definition of variables, and a list of determinant variables. The complexity of the second part of the algorithm is :

Proposition 5.2 *Let n_s be the number of states in the EFSM of the specification, $|R(x_i)|$ the number of values the variable x_i can take (in the interval of definition), and n the number of variables, then there is in $O(n_s(\prod_i |R(x_i)|)(2^n - 1))$ possible configurations.*

We must balance this complexity with the power of the algorithm. The worst case of this algorithm is the case where there is an error because we must check every path of the past. When there is no error the backward checking algorithm gives a sure answer (in constrast with former algorithms) at the first correct path we meet (that is supposed to be fast using the transition choice strategy). Anyway, the backward checking - if we consider only the trace analysis - is an improvement of former algorithm, and has the same complexity.

5.4. Gain in complexity

Here, we present the maximal complexity of our parallel algorithm.

Obviously, concerning the trace itself the complexity of normal and parallel algorithms are comparable. They are both proportional to the length of the trace. The difference is that the normal algorithm processes a transition in N_t where N_t is the number of transitions holding a given event couple, while the parallel algorihtm does it in 1.

With regards to the past of the trace, the complexity of the parallel algorithm is much lower. Actually, its maximal complexity depends on the number of states in the EFSM because the longest path without loops between two states goes through all the other states once. Nevertheless, a loop, in other words the path from a state to the same state through a non empty sequence of transitions, does not imply that the two considered configurations are equal. So the complexity also depends on the values of the variables, and more specially of the one with the largest definition interval.

Proposition 5.3 *Let $|R(x_i)|$ be the number of values the variable x_i can take (in the interval of definition), and N_s the number of states in the EFSM. The complexity of the parallel algorithm is then given by the following fomula :*

$$max|R(x_i)|.N_s$$

This result is very important because it makes from the parallel backward checking algorithm the first ever made linear algorithm for exhaustive error detection on EFSM. Then a total real-time error detection is possible, and its applications in terms of implementation correction and also in certain cases in Intrusion Detection System (IDS) is fundamental (cf. [10]).

6. Application examples

In this section we propose to illustrate the theoretical advantages of the parallel algorithm. For this purpose we introduce four network protocols with various characteristics. Two of them are small ones : Simple Connection Protocol (SCP), and Initiator-Responder Protocol (INRES). The third, Optimized Link State Routing Protocol (OLSR) is used in ad-hoc networks, and the fourth is the Open Shortest Path First Protocol (OSPF) used in wired networks.

Then we present the compared performances of the algorithms, emit remarks and conclude.

6.1. SCP

SCP allows us to connect an entity called *upper layer* to an entity called *lower layer* after a negociation of the Quality of Service desired for the connection. SCP layer is between the two layers and has a role of intermediary for the connection establishment. The upper layer communicates the desired QoS to SCP layer and SCP layer gives three tries to the lower layer to accept the QoS. If the three tries fail, the upper layer must reinitiate the protocol. Elsewise, a connection is open between the upper and lower layer, and they can exchange data. An EFSM specification of this protocol can be found in [3].

6.2. INRES

The INRES System is a simplified service and protocol - as SCP - that tries to establish a connection followed by a data exchange between two processes : an initiator and a responder. We consider in the study the EFSM of the initiator process as described in [5]. In this system, the initiator sends a first message and waits a limited time after which it must reinitiate the protocol. If it receives the appropriate message, the connection is established and the two process can exchange messages.

6.3. OLSR

The OLSR protocol [11] is a link-state proactive protocol designed specifically for mobile ad-hoc networks. OLSR manages to diffuse routing information through an efficient flooding technique. The key innovation of this protocol is the concept of Multi Point Relays (MPRs). A node multipoint relay is a subset of its neighbors whose combined radio range covers all nodes two hops away. In order for a node to determine its minimum multipoint relay set based on its two-hop topology, periodic broadcasts are required. Similar to conventional link-state protocols the link information updates are propagated throughout the network. However, in OLSR when a node has to forward a link

update it only forwards it to its MPR set of nodes. Finally, the distribution of topological information is realized with the use of periodic topology control messages and has as a result each node knowing a partial graph of the topology of the network that is further used to calculate the optimal routes. An EFSM of this protocol is available in [10].

6.4. OSPF

OSPF [12] is a very widely spread intra-domaine routing protocol using the Open Shortest Path First Algorithm. An OSPF Neighbour State Machine is used to maintain connections between two OSPF neighbour routers, and to exchange information on the link state through Link State Advertisement (LSA). The variables, like sequence numbers, are used for recording the present connections state. Such an EFSM is presented in [6].

6.5. Compared Performances

Given the EFSM of the pre-cited protocols, we can compare the performances of the different algorithms on them. To have a clear view of the results we sum them up in the following tables. N_s is the number of states, N_v the number of variables, V number of values that each variable can take, C the complexity, L_i the largest variable interval, and N_p the number of required processors.

protocol	N_s	N_v	V	C
SCP	4	4	4; 4; 4; 4	15360
INRES	4	3	5; 6; 2	1680
OLSR	4	6	2; 2; 2; 2; 5; 5	100800
OSPF	8	8	1; 2; 2; 2; 2; 2; 2; 2	261120

Figure 3. Complexity of the Sequencial Algorithm

protocol	L_i	N_s	N_p	C
SCP	4	4	8	16
INRES	6	4	28	24
OLSR	5	4	23	20
OSPF	2	8	88	16

Figure 4. Complexity of the Parallel Algorithm

In the two tables we present respectively all the characteristics needed to compute the complexity of the non-parallel and parallel algorithms. The figures in the tables last columns are not in seconds nor milliseconds but in transition processing time. It allows us to compare the results and can be abstracted to whichever unit of time. For instance we can read from these tables : if we need 261120ms (more than 4 minutes) to finish a trace analysis from OSPF with sequential algorithm, we need only 12ms with the parallel algorithm.

There are few other details to be explained about the tables data. For the number of values that a variable can take we made a simplification in two cases :

- in the INRES protocol one variable is not bounded (old_data) but as it is used only for a simple comparison (no intern modification) we can abstract it to a boolean;

- in the OSPF two variables are defined on the IP address domain ($[0;\ 2^{32} - 1]$) but are used for simple comparison, so we abstracted them too.

We must note that the parallelization implies the use of multi-processors architectures. Although there is no problem to organize a cluster of machines in wired world, it isn't the same situation for wireless networks in terms of bandwidth and security. The only valid possible parallel architecture for wireless protocols would be a massively multi-processor machine treating the protocol and its control through our parallel algorithm at the same time. For instance, it would need a 23 processors portable machine to control OLSR in real situation, and that is the reason why the OLSR example is nowadays not experimentally possible.

If we compare the protocols we can see that there is no doubt about the gain the parallel algorithm brings : it is 70 (INRES) to more than 20000 (OSPF) times faster and it processes the four protocols with an average of less than 20 transitions process, which is very low for an exhaustive analysis.

However, we can wonder when we look at the INRES protocol. Indeed, its complexity is 10 times lower than the SCP's one for the non-parallel algorithm and 1.5 time bigger for the parallel one. In addition, it needs even more processors than OLSR which is commonly considered as more complex. This can be explained by the fact that SCP, INRES, and OLSR are comparable in terms of number of states, but INRES uses variables that allows more values. In practice, the limitating factor tends to be the number of values allowed to the variables because EFSMs of real protocols are rarely composed of more than a ten of states.

7. Conclusion and future work

In this article we reminded the backward checking approach for passive testing, and its limitations. Then, we proposed our contribution consisting in few methods of parallelization of the algorithm and the study of each of these methods in terms of complexity and performance. Finally we have shown the undeniable power of the parallel version illustrated by the application to four well known and used communication protocols. This study has proven the linear complexity of the parallel algorithm, then making from it the first ever linear exhaustive algorithm for passive testing, and a serious mean of real-time monitoring for real life protocols and services.

Nevertheless, the application examples show the expected theoretical results. In other words, it would be a priority, in a future work, to implement the complete system architecture and evaluate the impact of different facts such as the parallel schedule management or the synchronizations, not taken into acount in this paper. It could be also interesting to elaborate an incremental parallel backward checking which could avoid repetitions in the computing process. This point is left for study in a future work.

References

[1] B. Alcalde, A. Cavalli, D. Chen, D. Khuu, D. Lee, *Network Protocol System Passive Testing for Fault Management - a Backward Checking Approach*, Lecture Notes on Computer Science, vol. 3235, pages 150-166, Springer, 2004.

[2] J.A. Arnedo, A. Cavalli, M. Núñez, *Fast Testing of Critical Properties through Passive Testing*, Lecture Notes on Computer Science, vol. 2644/2003, pages 295-310, Springer, 2003.

[3] A. Cavalli, C. Gervy, S. Prokopenko, *New approaches for passive testing using an Extended Finite State Machine specification*, in Journal of Information and Software Technology 45(12) (15 sept. 2003), pages 837-852, Elsevier.

[4] R. Hao, D. Lee, and J. Ma, *Fault Management for Networks with Link-State Routing Protocols* Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS), April 2004.

[5] D. Hogrefe, *Report on the Validation of the INRES System*, Technical Report IAM-95-007, Universitat Bern, November 1995.

[6] D. Lee, D. Chen, R. Hao, R.E. Miller, J. Wu and X. Yin, *A formal approach for passive testing of protocol data portions*, Proceedings of the IEEE International Conference on Network Protocols, ICNP'02, 2002.

[7] D. Lee, A.N. Netravali, K. Sabnani, B. Sugla, A. John, *Passive testing and applications to network management*, IEEE International Conference on Network Protocols, ICNP'97, pages 113-122. IEEE Computer Society Press, 1997.

[8] http://www.llnl.gov/computing/tutorials/parallel_comp/

[9] R.E. Miller, and K.A. Arisha, *On fault location in networks by passive testing*, Technical Report #4044, Departement of Computer Science, University of Maryland, College Park, August 1999.

[10] J.M. Orset, B. Alcalde and A. Cavalli, *An EFSM-based Intrusion Detection System for Ad Hoc Networks*, ATVA'05, October 2005.

[11] T. Clausen and P. Jacquet, *IETF RFC 3626 - Optimized Link State Routing Protocol (OLSR)* , The Internet Society, October 2003.

[12] J. Moy, *IETF RFC 2328 - OSPF Version 2* , The Internet Society, April 1998.

[13] M. Tabourier and A. Cavalli, *Passive testing and application to the GSM-MAP protocol*, in Journal of Information and Software Technology 41(11) (15 sept. 1999), pages 813-821, Elsevier, 1999.

[14] B. Tork Ladani, B. Alcalde and A. Cavalli, *Passive Testing - a Constained Invariants Checking Approach*, Lecture Notes on Computer Science, vol. 3502, Elsevier, 2005.