Teaching Distributed Algorithms Using SPIN

Claude Jard[1]

Université Européenne de Bretagne
IRISA/ENS de Cachan
Campus de Ker-Lann, 35170 Bruz, France

Abstract

This paper presents a teaching experiment to introduce formal methods and distributed algorithms at the undergraduate level in the department of computer science and telecoms of the ENS de Cachan in France. The course intends to teach some basic notions on formal modeling and analysis of distributed software. We used the free-software SPIN. This kind of experiment was found to be useful for our students since they gain an understanding of the importance and necessity of formal methods for designing even simple protocols.

Keywords: Distributed computing, Formal methods, Model-checking, SPIN, Promela.

1 Context

1.1 The ENS de Cachan in France

ENS de Cachan is a prestigious public institution of research and higher education, founded in 1912. It is one of the French "Grandes Ecoles" which are considered to be the pinnacle of French higher education. Students who enter the ENS have a double status: they are both students, registered in State Universities, and "normalien", i.e. members of the ENS. ENS de Cachan has two campuses, one located in Cachan near Paris, and one locate in Rennes in Brittany (West of France). Admission is decided by a highly selective national competition usually taken after a two-year post-baccalauréat preparatory program. The first year of education corresponds to the international BA/BS bachelor degree. Our students enter then in a Master program, before becoming a PhD candidate for most of them. They stay four years in the school.

1.2 Computer Science and Telecommunication department

This recently created department (2003) is devoted to Communication and Information Technology and Sciences. Its goal is to develop training and research in

[1] Email: Claude.Jard@bretagne.ens-cachan.fr

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

computer science, on the border of telecommunication. It is located in the Brittany extension of the ENS de Cachan on the Ker-Lann campus, in the south of the city of Rennes. The curricula are organized within the scope of the Master's degree entitled "Computer Science and Telecommunication" co-delivered by ENS de Cachan and the IFSIC Institute at the University of Rennes 1.

In addition to the usual set of degrees offered at IFSIC (Higher Training Institute for Computer Science and Communication), Department students are offered complementary courses in the fields of mathematics applied to statistics and signal processing, electronics and optics. But above all, these future researchers are offered personalized coaching, to help them maturing their taste for research and their professional objectives. For instance, this includes personal tutoring by professional academic staff, seminars, team work, internships in France and abroad, and, most important of all, a daily contact with researchers. We train 15 students each year, among which a dozen "normaliens" from ENS de Cachan and a few university students selected on the basis of their qualifications and by interview.

Telecommunications have been a leading research and development activity in Brittany for a long time. The Department is closely associated with the IRISA Research Institute (www.irisa.fr), a Joint Research Unit between CNRS, INRIA, University of Rennes 1 and INSA of Rennes. IRISA includes more than 180 permanent academic research staff, and as many PhD students. Together with the technical and administrative support staff, this sums up to more than 500 people. This makes IRISA one of the largest Computer Science laboratory in France.

2 The course on distributed algorithms. Its objectives.

The course takes place in the first semester of the first year of the school. It is a specific course offered to the department students. It is clearly their first approach of the area of distributed computing. This is quite unusual, but we found important to introduce the parallel and communication aspects of computer systems in the same time as the traditional concepts of sequential systems. After all, distributed computing is at the heart of modern systems and we are in a Computer Science and Telecoms department... Distributed computing means designing and implementing programs that run on two or more interconnected computer systems. For this level of course (undergraduate), we are assuming students will study distributed programs for a fully functioning Internet [5]. The intent is to use a network; not to study networks (this will be taught in the second year). Modern distributed programming includes multimedia systems, client-server systems, web programming and collaborative systems. What common fundamental principles and difficulties do they possess that will serve our students' needs for the next ten years?

Our students have been mainly selected on a basis on an excellent background and performance in mathematics. When there were recruited, they had the choice to follow a degree course in maths or in computer science. Maths in France are clearly the most prestigious way and those who decided to join the computer science track (by interest or realism, considering the possible opportunities of carrier) have to manage this frustration. This context benefits to the formal methods. A challenge for the teachers is to prove that computer science can be rigorous and

mathematically based. The situation is different in standard universities in France, where students often choose Computer Science against Mathematics to learn more practical things.

The last point is that we found distributed computing is an excellent playground to start a research oriented activity based on small collective projects. Distributed algorithms remain small, but presents surprisingly complex and counter-intuitive behaviors. This makes the students strongly aware of problems of software engineering.

To sum up:

- our context is favorable of the use and promotion of formal methods,
- we chose to start directly by teaching some aspects of distributed computing at the early stage of the degree course in computer science,
- it is a good place for research-oriented activities, and
- the intrinsic complexity of even small distributed algorithms is a good carrier of the promotion of formal methods in the context of a mathematically based software engineering.

From the algorithmic point of view, the objective of the course is to put emphasis on the difficulty of designing correct distributed algorithms. The study of several classical paradigms (reliable transfer, mutual exclusion, termination) serves to design a general methodology of description. The course is divided into six sessions of two hours long. It also comprises several projects, conducted by groups of 2-3 students.

3 About the use of SPIN

3.1 SPIN and Promela

SPIN [4] is a popular open-source software tool, used by thousands of people worldwide, that can be used for the formal verification of distributed software systems. The tool was developed at Bell Labs in the original Unix group of the Computing Sciences Research Center, starting in 1980. The software has been available freely since 1991, and continues to evolve to keep pace with new developments in the field. In April 2002 the tool was awarded the prestigious System Software Award for 2001 by the ACM. Since 1995, (approximately) annual SPIN workshops have been held for SPIN users, researchers, and those generally interested in model checking. SPIN is an automata-based model checker. Systems to be verified are described in Promela (Process Meta Language), which supports modeling of asynchronous distributed algorithms as non-deterministic automata. Properties to be verified are expressed as Linear Temporal Logic (LTL) formulas, which are negated and then converted into Buchi automata as part of the model-checking algorithm. In addition to model-checking, SPIN can also operate as a simulator, following one possible execution path through the system and presenting the resulting execution trace to the user. Unlike many model-checkers, SPIN does not actually perform model-checking itself, but instead generates C sources for a problem-specific model checker. This rather antique technique saves memory and improves performance, while also al-

lowing the direct insertion of chunks of C code into the model. SPIN also offers a large number of options to further speed up the model-checking process and save memory.

Promela programs consist of processes, message channels, and variables. Processes are global objects that represent the concurrent entities of the distributed system. Message channels and variables can be declared either globally or locally within a process. Processes specify behavior, channels and global variables define the environment in which the processes run.

3.2 Selected usage

Assuming our students are familiar with a standard sequential imperative programming language like C, the best originality of Promela is its non-deterministic control flow constructs if (selection command) and *do* (repetition command), based on the concept of guarded commands introduced by Dijkstra [2]. The selection is a list of guarded commands, of which one is chosen to execute. If more than one guards are true, one statement is non-deterministically chosen to be executed. If none of the guards are true, the result is undefined. Because at least one of the guards must be true, the empty statement "skip" is often needed. The repetition executes the guarded commands repeatedly until none of the guards are true.

This type of abstract construct is a bit disconcerting for the students. On one hand, specifying behaviors in Promela is a good opportunity to explore the existing continuum between the different abstraction levels, which are concerned with modeling activities, from the code level to the possible logical level used to express properties. A crucial question in formal methods is actually to choose a relevant level of abstraction.

On the other hand, I must admit that a graphical representation like the communicating finite state machines of Figure 1 is better suited for a pedagogical presentation on the blackboard. In practice, I used automata, even extended automata to take into account variables and symbolic conditions to model more complex distributed algorithms. The translation to Promela was based on a systematic way to code communicating automata into Promela programs (see the code of Figure 2).

The other graphical aspect is to present the execution traces and the different examples and counter-examples. It is quite easy to use informal message diagrams on the blackboard in exact correspondence with the message sequence charts displayed by SPIN (see Figure 3).

Once the Promela program is written, the interactive simulator is used to validate the code and we discuss how to instrument the code to perform verification. The course is not a course on verification. It just explains the general idea of exhaustive simulation provided by the enumeration of the possible reachable states. It is clearly out of scope at this level to present the temporal logic, its translation into Buchi automata and the on-the-fly construction of the graph product, on which the satisfaction of properties is computed. We thus preferred to use the simple method of assertions, which can be easily inserted into the Promela code, at the price of inventing special global variables to compute the invariants. The difficulty is to explain that these variables are not part of the model, but are just there for

verification purpose.

SPIN possesses the interesting feature to detect violations of assertions and when this occurs, to graphically present the shortest scenario leading to the violation in term of message sequence charts. This was quite impressive for the students, since we had in a few seconds for our small examples the discovering of counter-examples for various conjectures difficult to reject by hand. It was also a good example to see how computers may help the design of other computers despite the theoretical barrier of computability, seen in another course in parallel.

To sum up:

- the presentation of the algorithms is based on a graphical representation on the blackboard as well as the execution traces, shown as message diagrams.
- the formalization is completed using a translation towards Promela and the insertion of global assertions,
- the ability of SPIN to rapidly present non-trivial counter-examples is impressive and proves the interest of automated tools in that case.

4 Introduction using the simplest protocol

We start the course by introducing the simplest protocol that we can imagine between two processes A and B communicating asynchronously through reliable FIFO queues. I present the problem of modeling the interaction between a user A and a timer B. The user has only two states: it can be awoke or sleeping. When awoke, it can ask to set the timer (message a) and decides to sleep. After a certain period of time (not modeled here and abstracted as a spontaneous transition), it can either wake up and ask for the stop of the timer (message b), or it is awoke by a timeout ringing (message c), sent by the timer process. Symmetrically, the timer process has two states: timer set or not. The timer is set upon receiving the set message. It is stopped when it decides to send the timeout message or when it receives the stop message. Messages from A to B are stored in a FIFO communication channel linking A and B, while the messages in the other direction are stored in a second FIFO communication channel linking B and A. In case of bounded channels, we consider that sending a message on a full channel will block the communication until a possible reception. The protocol is explained using the formalism of communicating automata illustrated in the left part of Figure 1.

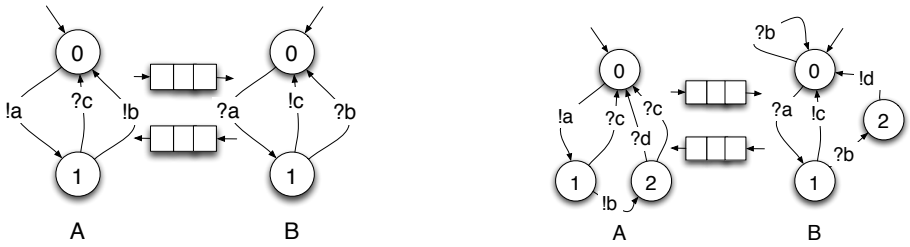

Fig. 1. Our first protocol. Left part: a buggy version. Right part: a corrected one.

The translation into Promela is shown on Figure 2. Notice the introduction of the global variable *alternate* to check that it is not possible to have two consecutive

message a in the communication channel from process A to process B. My favorite joke is to prove informally that is not possible: the only mean to put a second a in the channel is to execute again the transition from state 0 to 1 of the process A. And this implies for it to go back to state 0, which demands for the process B to consume the first message a. This is false, but the shortest counter-example needs to consider two cycles in the protocol with an asynchronous collision between messages b and c in the channels. This happens at depth 8 in the state graph, which is not easy to guess by hand. Figure 3 shows a screen shot of this invalidation. We concluded that distributed algorithms are complex objects, since even for the simplest one, it is difficult to obtain an evidence of correctness. A corrected version is proposed as shown in the right of Figure 1, where a supplementary message d is introduced to acknowledge the stop of the timer.

```
/*-------------------------------------------------------------------------*/
/* My favorite first example of complex behaviors in distributed systems.
   Buggy version including assertion */
#define N 8 /* Size of channel AB */
mtype = {a,b,c}; chan AB = [N] of {mtype}; chan BA = [1] of {mtype};
bit alternate = 1;  /* For verification purpose */

active proctype A() {do
 ::  atomic{ assert(alternate); alternate = 0; AB!a };
 if :: BA?c; :: atomic{ AB!b; alternate = 1 } fi od}

active proctype B() {do
 :: atomic{ AB?a; alternate = 1 };
 if :: AB?b; :: BA!c fi od}
 /*-------------------------------------------------------------------------*/
```

Fig. 2. The wrong Connect-Disconnect protocol expressed in Promela

Exhaustive simulation is a good method to find bugs. Of course, it suffers from the state explosion phenomenon. It is not the sole viable approach for analyzing systems, especially when tempting to obtain a formal proof of correctness. It is easy to discuss this aspect with the students on the erroneous version of the protocol, which has an infinite state space. We were lucky that the situation with two consecutive a in the AB communication channel was occurred when bounding the channel by three messages (bounded channels are required by SPIN, which is based on an enumerated model-checking). In general, one cannot assert that bounding channels by a given value is enough to check a given property. This is undecidable, even for our simple model of two communicating finite state machines. It is interesting to note that we can predict in our small example the number of states in function of the size n of the AB channel, using the formula $\lfloor \frac{n^2}{4} \rfloor + 3n + 3$. This formula has been used to test several model-checker in the world.

At this point of the course, it is possible to try to build a formal proof of correctness of the "corrected" protocol. This rises the question of how to express the desired properties. One possibility is to use the temporal logic framework provided by SPIN. As I explained above, it was not realistic to implement the idea until now, since their first course on logic was put after my presentation. But this could be changed next year, and will offer a good opportunity to show an interesting application of logic. In Linear Temporal Logic, the property of the alternation of message a can be written as $\Box[(!a \wedge \Diamond\bigcirc !a) \Rightarrow \bigcirc\neg(\neg(!b \vee !c)\mathcal{U}!a)]$.

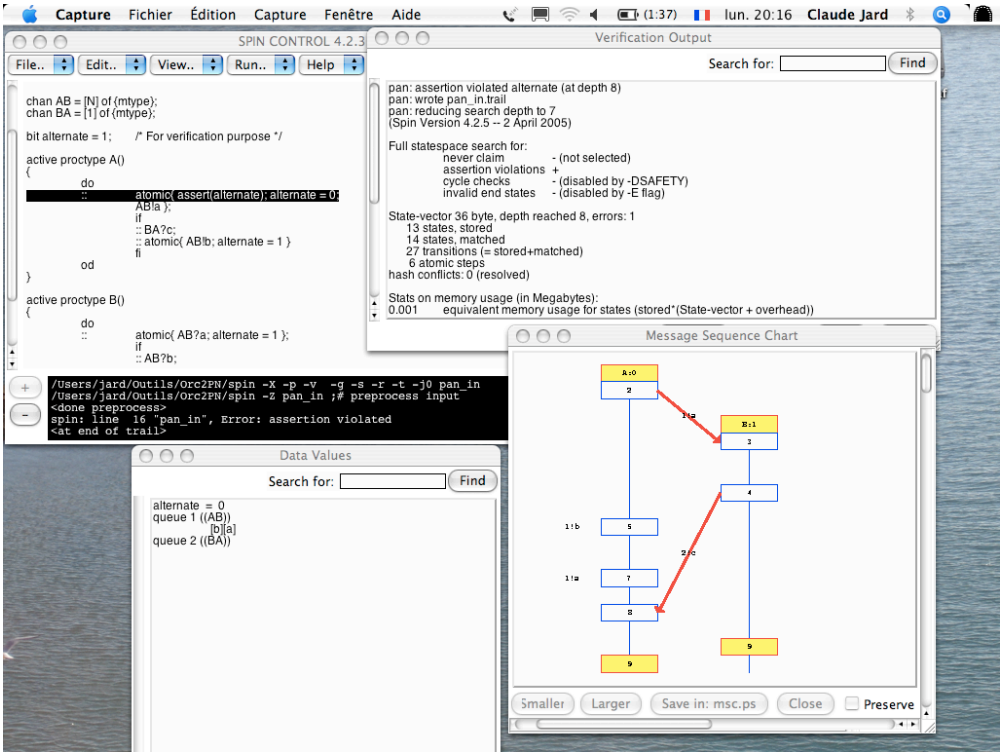

Fig. 3. Screen shot of the Spin session discovering the shortest path to the error.

Another kind of formal proof has been discussed during the following SWP project, in which some liveness property can be proved using transition invariants in a Petri net model.

5 The SWP project

The next step of the course is a homework project, based on the study of a simplified transport protocol, supposed to represent the essence of the Internet TCP. This "Stop and Wait Protocol" was described in [1], an invited paper in a conference on formal methods in 2003. Stop-and-Wait protocols have been shown to operate correctly over media that may lose packets, however, there has been little discussion regarding the operation of these protocols over media that can re-order packets. The model is analysed using a combination of hand proofs and automatic techniques based on the analysis of Coloured Petri net models. After a collective discussion to explain the content of the paper, present the Petri net model and the associated proofs, the goal of the project is to obtain similar results using SPIN. The results are the following:

- A proof of the counter intuitive property for a Stop-and- Wait protocol that the number of packets that are stored in the network can grow without bound. This is

true for any positive values of the maximum sequence number and the maximum number of retransmissions.

- It is shown that the protocol may fail: loss of packets is possible and duplicates can be accepted as new packets by the receiver, even though the sender and receiver perceive that the protocol is operating correctly.

```
/*-------------------------------------------------------------*/
/* "Stop and Wait protocol" from J. Billington, Forte'2003.
 *  A simplified transport Protocol after Internet/TCP */
#define true 1
#define false 0
#define empty 0
#define MaxData 2
#define MaxRetrans 1
#define MaxChannel 3           /* 2*MaxRetrans+1 */
#define MaxSeqNo 1
chan mess_channel = [MaxChannel] of {int,int}; /* Sender->Receiver (no_seq, data) */
chan ack_channel  = [MaxChannel] of {int};       /* Receiver->Sender (no_seq) */

active proctype Sender() /* Sender automaton */ {
 bit sender_ready = true; /* Two states automaton */
 int seq_no = 0;             /* Sequence numbering of data */
 int retrans_counter = 0; /* Retransmission counter */
 int data = 0;                 /* VERIF: the transmitted data */
 int m;                          /* Last number of the received ack */
 do :: atomic{sender_ready && data<MaxData -> data++; /* send_mess */
                printf ("MSC: Send request: %d\n", data);
                mess_channel!seq_no,data; sender_ready = false; }
     :: atomic{(retrans_counter<MaxRetrans) && !sender_ready -> /* timeout_retrans */
               mess_channel!seq_no,data; retrans_counter ++; }
     :: atomic{ack_channel?m -> /* receive_ack */
        if :: (m==(seq_no+1)%(MaxSeqNo+1)) && !sender_ready -> retrans_counter = 0;
           seq_no = m; sender_ready = true;
           :: (m!=(seq_no+1)%(MaxSeqNo+1)) -> skip; /* receive_dup_ack */ fi; } od}

active proctype Receiver() /* Receiver automaton (one single state) */ {
 int rec_no = 0; /* Sequence number of received messages */
 int rec_indication = 1; /* VERIF : Test of non detection of duplicated acks */
 int data; /* Last received data */
 int sn;    /* Its sequence number */
 do :: mess_channel?sn,data  -> /* receive_mess */
         if :: (sn==rec_no) -> rec_no = (rec_no+1)%(MaxSeqNo+1);
               printf("MSC: receive indication : %d\n",data);
               assert (data==rec_indication);/* VERIF */
               rec_indication++; /* VERIF */
            :: (sn!=rec_no) -> skip fi;
        ack_channel!rec_no; od}

active proctype devil () /* Simulation of the network level
                            (lossy channels with possible reordering) */ {
 int s,d;
 do :: atomic{mess_channel?s,d -> skip; } /* Loose message */
      :: atomic{(len(mess_channel) > 1) -> mess_channel?s,d; mess_channel!s,d; }
                                          /* Reorder message */
      :: atomic{ack_channel?s -> skip; } /* Loose ack */
      :: atomic{(len(ack_channel) > 1) -> ack_channel?s; ack_channel!s; }
                                          /* Reorder ack */ od}
/*-------------------------------------------------------------------------*/
```

Fig. 4. The Stop and Wait protocol expressed in Promela

The motivating fact is that it seems these failures (based on the problem of the incrementation of the sequence numbers using a finite modulo) could be reproduced in the real TCP, by a particular positioning of its large set of parameters.

Figure 4 shows the complete Promela program we obtain. We can notice the introduction of the "Devil" process to simulate losses and reordering of messages in the communication channels. As previously, some global variables have been introduced to be able to write assertions. To this goal, we maintain the exact numbering of data, to be able to check the correct sequence of receptions at the user level. In the screen shot of Figure 5 we detect a shortest counter-example in the case of reordering and for a small modulo of 2. Without reordering, as expected,

SPIN does not detect faults during verification, which can be performed only for small values of the different parameters because of the classical state explosion problem.

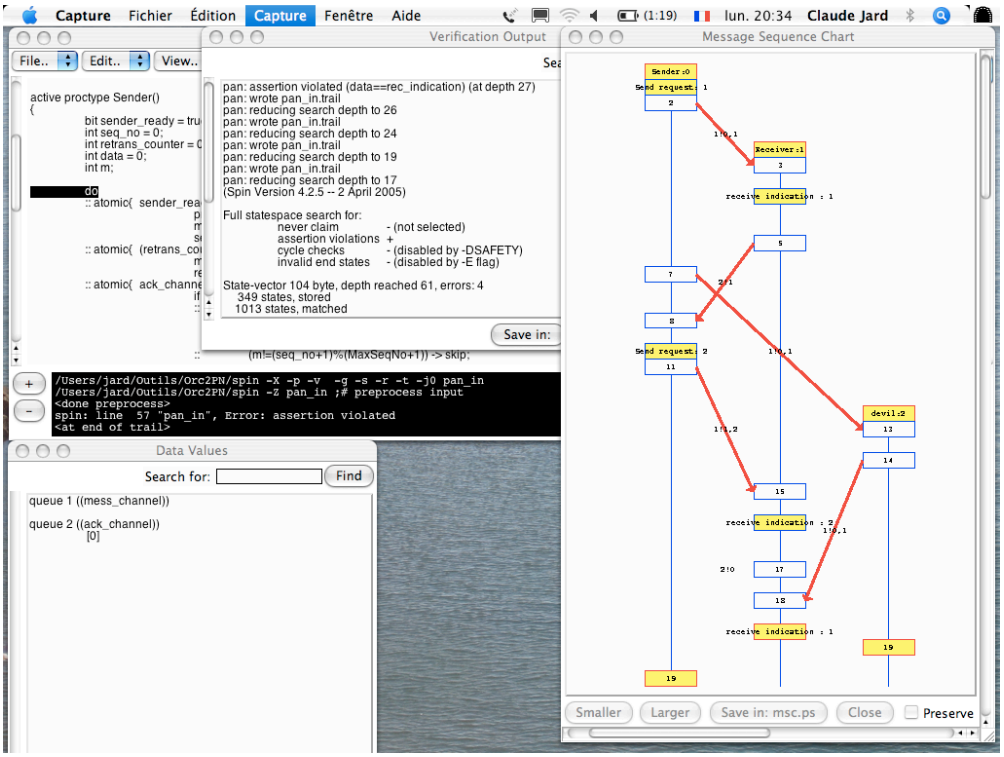

Fig. 5. Screen shot of the Spin session discovering the shortest path to the error.

The different models and results obtained by the students were presented during a short oral presentation (project defense), followed by a general discussion about what they have learnt.

The rest of the course, using a similar approach, is dedicated to the study of other basic distributed algorithms. For the problem of termination detection, I chose the Dijkstra's algorithm using a ring of processes. In this algorithm, a colored token circulates on the ring. If the token comes back to the initiator without having changed its color, the termination can be claimed. One can show it is correct in an asynchronous environment (recall that the original algorithm was designed for synchronous communication) under the assumption that no application message can be delayed more than the total time needed for the token to complete its tour on the ring. SPIN was able to produce a counter-example in the latter case.

The last part of the course presents the consensus problem and mainly the result of Fisher, Lynch and Paterson on the impossibility of achieving a consensus in presence of process failures [3].

6 Conclusion

6.1 What do the Students Learn ?

At the beginning of the course, our students are almost unaware of distributed computing and formal description techniques. Experiments using cases studies motivate them because they see it "with their own eyes". The main lesson is that it is difficult to understand the behavior of a protocol without the help of formal tools. Even for a simple protocol, they understand that automatic tools are necessary in order to detect errors. We expect that they will remember that techniques exist and hopefully that they will convince more people about their usefulness. The evaluation of the understanding of students is done throughout the case studies and the presentation of their home-works.

6.2 Future evolutions

I want to improve the practical aspect of the work in order to enforce the proof of potential impact of this kind of formal approach. To that respect, the SWP example is promising. The idea is to try to use the possible failures found on the simple model to force real TCP to make a mistake. From the application point of view, a duplication that is not detected may have spectacular effects (imagine a banking application for example). A realistic experiment is to use the NS simulator in which most of the TCP parameters can be changed. A more ambitious project is to settle a real experiment on the Internet. But before that, we have to evaluate if the parameters of our network environment make the fault possible (if it is the case, the probability of occurrence will be very low in practice of course).

References

[1] Billington, J., Gallash, G. E., "How Stop and Wait Protocols Can Fail over the Internet", Formal Techniques for Networked and Distributed Systems - FORTE 2003, Lecture Notes in Computer Science, Volume 2767/2003, pp. 209-223.

[2] Dijkstra, E., "Guarded commands, non-determinacy and formal. derivation of programs", Commun. ACM 18 (1975). http://www.ccs.utexas.edu/users/EWD/ewd04xx/EWD472.PDF.

[3] Fisher, M. J., Lynch, N. A., Paterson, M. S., "Impossibility of Distributed Consensus with One Faulty Process", Journal of the Association for Computing Machinery, Vol. 32, No. 2, April 1985, pp. 374-382.

[4] Holzmann, G. J., "The SPIN Model Checker: Primer and Reference Manual", Addison-Wesley, 2004. http://spinroot.com/spin/whatispin.html.

[5] Hyde, D. C., "Importance of Teaching Cluster Computing", Proceedings of the 8th International Parallel Processing Symposium, IEEE Computer Society Press, 1994, pp. 956-961.