



University of Glasgow | School of
Computing Science

PROBABILISTIC MODELLING AND ANALYSIS OF A NON-REPUDIATION PROTOCOL

IGHOROJE LAMOGHA

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in part fulfillment of the requirements
of the Degree of Master of Science at the University of Glasgow

2nd September, 2013


Abstract

Non-repudiation of a communication process means that the parties involved ‘cannot deny’ taking part in all or part of the communication process. Non-repudiation is important because it ensures and guarantees that the message transferred has been sent and received by both parties claiming to have sent and received them. It also helps to boost the confidence level/trust profile of the parties involved in a communication process. Protocols which offer non-repudiation services are referred to as non-repudiation protocols. Evidences generated during an execution of a non-repudiation protocol without a trusted third party are referred to as non-repudiation evidences (either a non-repudiation of origin or recipient evidence). Different models have been made of the probabilistic non-repudiation protocol without a trusted third party as proposed by Markowitch & Roggeman. Not having a trusted third party removes communication bottleneck, improves performance and security. However, with the intent of designing a realistic model that puts into consideration the behaviour of the parties involved, two problems of unfairness were discovered that puts the originator (provider of a service) at a disadvantage. The first problem is that the recipient (receiver of a service) always gets his evidence of non-repudiation first. Secondly the originator’s evidence is earned only when he receives the final acknowledgement while the recipient’s evidence is earned incrementally (a combination of all the messages received from the originator). Two solutions are considered that helps in improving the fairness of the protocol. The first is that both parties receive their non-repudiation evidence incrementally and the second is that they receive their evidence in an interchangeable manner. When analysed in PRISM, there was an improvement in the fairness of the protocol.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

Name: Lamogha Ighoroje

Signature: 

Acknowledgements

Thanking God almighty, my parents and my supervisor for their help and support and making my dreams come true.

Contents

Chapter 1	Introduction.....	1
Chapter 2	Survey.....	5
2.1	The Protocol.....	5
2.1.1	Steps of the protocol execution.....	5
2.1.2	Properties and requirements of the protocol	6
2.2	Probabilistic model checking	6
2.3	PRISM model checker:.....	7
2.3.1	The PRISM Language.....	7
2.4	Discrete Time Markov Chains (DTMCs)	8
2.4.1	Property specification for a DTMC	10
2.5	Related Works:.....	11
Chapter 3	Requirements for Model Design	15
3.1	Discussion of problem found with regards to fairness	15
3.2	Intended improvements of fairness.....	16
3.3	Assumptions made	17
3.4	Description of Models.....	18
Chapter 4	Modeling and Analyzing the Protocol (version 1)	20
4.1	The PRISM codes of the models explained.....	21
4.2	Properties and rewards considered	24
4.3	Analysing the properties of the protocol in PRISM.....	25
4.3.1	Probability that the protocol terminates correctly	25
4.3.2	Expected evidence each party gains	26
4.3.3	Probability of unfairness in the protocol.....	27
4.3.4	Time each party spent in an unfair state	27
4.3.5	Expected number of steps for the protocol run	29
4.4	Conclusion	30
Chapter 5	Accumulating Evidences for both Parties (Version 2)	31
5.1	Introduction.....	31
5.2	Analysis of Version 2.....	33
5.2.1	Probability of Version 2 terminating correctly	33
5.2.2	Expected evidence each party has after k steps in version 2	33
5.2.3	Probability of unfairness in version 2.....	34
5.2.4	Time each party spends in an unfair state in version 2	34
5.2.5	Expected time (number of steps) to complete the protocol run in version 2	35

5.3	Conclusion	35
Chapter 6 Evidence before Message at Last Step (Version 3).....36		
6.1	Introduction.....	36
6.2	Analysis of Version 3.....	38
6.2.1	Probability of version 3 terminating correctly.....	38
6.2.2	Expected evidence each party has after k steps in version 3	38
6.2.3	Probability of unfairness in version 3.....	39
6.2.4	Time that each party spends in an unfair state in version 3.....	40
6.2.5	Expected time (number of steps) to complete the protocol run in version 3	42
6.3	Conclusion	42
Chapter 7 Evidence before Message after Start (Version 4)43		
7.1	Introduction.....	43
7.2	Analysis of Version 4 in PRISM.....	44
7.2.1	Probability of version 4 terminating correctly.....	44
7.2.2	Expected evidence each party has after k steps in version 4	44
7.2.3	Probability of unfairness in version 4.....	45
7.2.4	Time each party spends in an unfair state in version 4.....	45
7.2.5	Expected time (number of steps) to complete the protocol run in version 4	47
7.3	Conclusion	47
Chapter 8 Interchangeable Exchange (Version 5).....48		
8.1	Introduction.....	48
8.2	Analysis of Version 5 in PRISM.....	51
8.2.1	Probability that version 5 terminates correctly	51
8.2.2	Expected evidence each party has after k steps in version 5	51
8.2.3	Probability of unfairness in version 5.....	51
8.2.4	Time each party spends in an unfair state in version 5.....	52
8.2.5	Expected time (number of steps) to complete the protocol run in version 5	55
8.3	Conclusion	55
Chapter 9 Conclusion and Future Work56		
9.1	Future Work.....	57
9.1.1	PTA Modeling and Analysis	57
9.1.2	Applying stochastic two player games.....	58
9.1.3	Detailed Security and Performance analysis.....	58
9.1.4	Extending the protocol to a Multiparty Scenario	58
Appendix A Prism Codes and simulation of models59		
A. Property Specification Codes.....		59
A.1	Version 1: PRISM codes and simulations.....	59
A.2	Version 2: PRISM code and simulation.....	62
A.3	Version 3: PRISM codes and simulation.....	69

A.4	Version 4: PRISM codes and simulations.....	77
A.5	Version 5: PRISM codes and simulation.....	85
Bibliography	94

Chapter 1 Introduction

Different models have been made of the probabilistic non-repudiation protocol without a trusted third party as proposed by Markowitch & Roggeman (1). In this chapter I introduce the subject matter of non-repudiation protocols and probabilistic model checking in general. The motivation for undergoing this project and an outline of the remainder of this report is also provided in this chapter.

A communication process involves the flow of information between an **originator** (*the provider of the information or service*) and a **recipient** (*the receiver of the information or service*). This usually involves different steps such as:

- The recipient asks the originator for a service.
- The originator provides the service.
- The recipient gives some evidence that he/she has received the service e.g. by making a payment or by giving a receipt (referred to as an acknowledgement in the electronic world).

A typical communication flow is seen in figure 1.

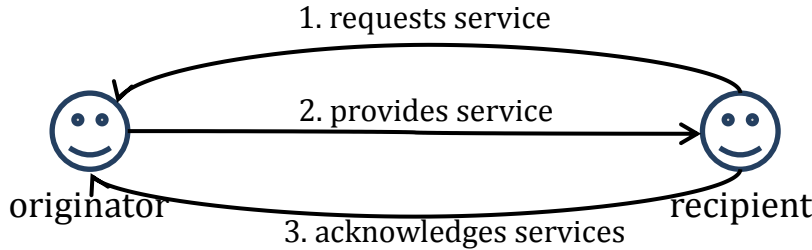


Figure 1: simple communication flow between an originator and a recipient.

Repudiation of a communication process is when a party (either the originator or the recipient) denies that they took part in all or part of the communication.

Therefore, **non-repudiation** of a communication process means, ‘not being able to deny’ taking part in all or part of the communication process (1). So the inability of the originator to deny providing the service is referred to as *Non-Repudiation of Origin (NRO)* and the inability of the recipient to deny that they received the service is referred to as *Non-Repudiation of Receipt (NRR)*. It is desired that, at the end of the communication process, the recipient and originator possess the NRO and NRR evidence respectively. Protocols which offer non-repudiation services are referred to as **Non-Repudiation Protocols (NRPs)**.

Non-repudiation is important because it helps to ensure and guarantee that the message transferred has been sent and received by the parties claiming to have

sent and received them. This helps to boost the confidence level of both parties about a communication process. The fact that there are NRO and NRR evidences, makes the originator confident that the recipient cannot later deny receiving the service. Likewise the recipient is confident that the originator cannot deny that they provided the service.

In many non-repudiation protocols, there is the need to have a *Trusted Third Party (TTP)* involved in the protocol's execution in order to have a non-repudiated communication between the two parties. An example of this type of protocol is seen in (2). The responsibility of the third party is to act as a delivery messenger and dispute resolver for the two parties.

Markowitch and Roggeman (1) aim to show that it is possible to achieve non-repudiation without the need of a TTP. Reasons for not having a TTP include:

- Improved performance and communication flow: as TTP interference may cause communication bottlenecks thereby limiting performance.
- Not having a TTP gives lower communication overhead.
- There is no fear of deciding whether the TTP abused privileges and intervened in the protocol execution without cause or need.
- The less people that partake in a communication process, the more straightforward and simple the process becomes.

In the protocol of Markowitch & Roggeman, they were able to eliminate the need of a third party by including two extra non-repudiation services which include: *non-repudiation of submission* (to ensure that the originator submits the message for delivery) and *non-repudiation of delivery* (to ensure that the message is delivered to the recipient). Markowitch & Roggeman (1) explain that the most frequently occurring problem is the issue of obtaining an acknowledgment from the recipient as evidence that they have received what they ought to receive.

There are certain properties that a non-repudiation protocol must or should have and these properties include:

A non-repudiation protocol has to be **fair**: this simply means that at any time the protocol is been executed, either both parties receive what they are expecting or neither party receives anything.

It has to be **bounded by time**: i.e. information transferred at each step of the protocol run should be executed within a specified time limit till the end of the protocol execution. The protocol can be stopped when the delay is exceeded and at least one of the parties is behaving correctly.

It should be **viable**: i.e. at the end of the protocol execution, if both parties have behaved correctly, then they receive what they are expected to receive.

To achieve fairness, the protocol proposed by (1) can be modelled in several ways to exhibit and analyse different types of behaviours such as:

- *Probabilistic behaviour*: there is a probabilistic choice as to the number of steps of the protocol.
- *Non-deterministic behaviour*: at each step during the protocol's execution the originator and recipient can choose between different possible behaviors. And since there are two parties i.e. the originator and recipient participating with different objective goals and choices, games can be used to analyze the protocol (3).
- *Real time behaviour*: there is a timing constraint on the system that has to be met.

Probabilistic model checking is a formal method of quantitatively checking systems that exhibits one or more of these behaviours.

And formal analysis of a system or process may involve and require different processes (3) such as:

- Designing the system models, through specifications and high level diagrammatic designs.
- Verifying the behaviour of a system's model. And verification requires model checkers (to check the authentication of statements in a state transition system).
- Checking whether two or more models can be combined to give a desired behaviour.
- Combining an implementation from a formal specification, such that all the behaviours in the specification are also present in the implementation.
- Checking whether one model simulates another (i.e. all behaviours of the latter is present in the first) or if two models are bi-similar (i.e. produce identical behaviour).

Over the years, there have been different models and analysis of the non-repudiation protocol proposed by Markowitch & Roggeman (1). Some of the models of the protocol considered so far in this project have been probabilistic in nature without non-determinism; some have been more concerned about performance, while some have modelled the protocol as a game in order to represent the different goals of the originator and recipient.

The major contribution of this project is the development and formal analysis of a concrete model that helps to improve fairness of the protocol proposed by Markowitch and Roggeman. This is because one common thing that non-repudiation protocols modelled must have is fairness. And with the intent of designing a realistic and fair model that puts into consideration the behaviour of the parties involved, two problems of unfairness was discovered that puts the originator (provider of the service) at a disadvantage. The first problem is that the recipient (receiver of the service) always gets his evidence of non-repudiation first. Secondly the originator's evidence is earned only when he receives the final acknowledgement while the recipient's evidence is earned incrementally (a combination of all the messages received from the originator).

Two solutions are considered that help in improving fairness of the protocol. The first is that both parties receive their non-repudiation evidence incrementally and the second is that they both get to receive their evidence first in an interchangeable manner.

PRISM is a probabilistic model checking tool used for the formal model verification of systems that exhibit random or probabilistic behaviours. Application areas of PRISM includes: communication and security protocols, biological systems, etc. It can build and analyse several types of probabilistic models e.g. discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs), Markov decision processes (MDPs), probabilistic automata (PAs) and probabilistic timed automata (PTAs) (4) (5). In addition, these models can be extended with costs and rewards to analyse some properties e.g. PRISM-games for example is a new extension of PRISM for probabilistic model checking of stochastic multi-player games. Since PRISM is a probabilistic model checking tool and has extensions to support modelling and analysis of the protocol's properties or parties behaviour, makes the PRISM tool a good model checking tool for this project.

For simplicity, it is assumed in this project that the communication channel between the originator and recipient is an operational channel. Therefore, no source information is lost during the transmission. This means there will be less focus on the channel of communication and more focus on modelling and analysing the fairness of the protocol.

The remainder of this dissertation report is structured as follows:

Chapter 2 of this report gives a literature survey about probabilistic modelling and analysis of communication protocols. The initial part of this Chapter focuses on the non-repudiation protocol as proposed by Markowitch and Roggeman in (1) which is the probabilistic protocol of interest in this project.

Chapter 3 considers the requirements needed to design the models and the assumptions made during the design process. The problem of the protocol with regards to fairness is discussed in this chapter as well with the intended approach on how fairness of the protocol can be improved.

Chapter 4 focuses on the design and analysis of the basic models (version 1 of the models) of the protocol as proposed by Markowitch and Roggeman. This was done to explain the problems of unfairness found.

Chapter 5, 6, 7 and 8 of this report focuses on modifications made to the models in Chapter 4 to give versions 2, 3, 4 and 5 of the models. An introduction highlighting the difference between each version and the previous version discussed is provided, a property analysis of these new versions in PRISM is given and a conclusion about the results obtained is provided.

Chapter 9 gives a final conclusion of the work done in this report, its limitations and suggestions of future works that can be achieved.

Note: Supplementary materials e.g. PRISM codes, documentation, detailed evaluation result (i.e. simulation of the models and graphs obtained) are submitted along with this dissertation.

Chapter 2 Survey

This chapter is based on the protocol model as proposed by Markowitch & Roggeman in (1). It also discusses probabilistic model checking, different probabilistic models and their properties in terms of the behaviors they exhibit. Then a brief description of the model checking tool that will be used i.e. the PRISM tool is given. Finally, a literature review of related works that have used probabilistic model checking for modeling and analysis of the protocol proposed by Markowitch & Roggeman is carried out.

2.1 The Protocol

Markowitch & Roggeman proposed in (1) a probabilistic non-repudiation protocol whose analysis is the basis of this project. The probabilistic non repudiation protocol proposed by them is one which does not require the need of a third party as discussed earlier in Chapter 1. The main steps of the protocol are described below. We assume the existence of an originator (O) who wants to send a message and a recipient (R) who wants to receive a message from O.

2.1.1 Steps of the protocol execution

R wants information from O therefore:

1. R determines the date **D** and makes the request.
2. O checks **D** and chooses the number of steps **n** for the protocol to be fully executed (**n** must be known to O alone).
3. Then O computes message functions for each step of the protocol.

The required **message** is a composition of all the functions.

$$\text{Message} = f_n (\text{message}) \circ (f_{n-1}(\text{message}) \circ (.... \circ (f_2 (\text{message}) \circ f_1 (\text{message})))...))$$

4. R chooses a key **k** to encrypt the message with.
5. O encodes each message function with the key, electronically signs it with date and time stamps and sends message S_m ($S_m = \text{encoding}(f_n(\text{message}), k)$).

Note: The originator sends the function (message) in a reverse manner, so that the recipient gets the last message **$f_n(\text{message})$** first. Since the required message is a composition of all the functions, it is ideal to use a composition operator e.g. \circ that is non-commutative to ensure that the recipient cannot start to compute the message until the final message $f_1 (\text{message})$ is received (1) (4). Functions must be mutually dependent on each other, so that even if the recipient successfully computes a function it would be meaningless to him unless he has every function that makes up the message.

6. At each step after receiving the message function from O, R is required to send an acknowledgement for that message function received. Each acknowledgment is electronically signed by the recipient with date, time and corresponding message number stamps.
7. Finally, at the end of the protocol execution, when the recipient has received all message functions, he can then decode the message with the key.

Note: at the end of a successful run of the protocol, it is expected that the originator has received the non-repudiation of receipt (NRR) evidence (this is usually obtained when he receives the last acknowledgement from the recipient) and that the recipient has the non-repudiation of origin (NRO) evidence which is obtained when he has received all the message functions and can go ahead to compute the message.

I.e. the NRR evidence equals $\text{Sign}_R(\text{ackn})$

While the NRO evidence equals $\{\text{NRO}_i \mid i = 1, \dots, n\}$, with $\text{NRO}_i = \text{Sign}_O(\text{fi}(\text{message}), O, R, D)$.

2.1.2 Properties and requirements of the protocol

At each step of the protocol, fairness can be achieved with a certain probability ϵ . The number of steps n required for the protocol to run is randomly selected according to a geometric distribution and is unknown to the recipient. Markowitch and Roggeman do not consider the behavior and computational power of the recipient (1). They assume that both parties behave honestly.

The recipient must not know when the last message is received, or else the tendency of not sending the last acknowledgment for that step is possible thereby making it easy for him to deny receiving the service (and this is not **fair** to the Originator). Also **deadlines** which are chosen and known by both parties **or by using an operational channel** are necessary to ensure that information transmitted is **bounded by time**. In addition, if an originator does not receive the acknowledgement from the recipient before the deadline, then he can stop the protocol run. If after the expected deadline for the recipient to get a message, he does not receive a message, he can assume that the protocol was terminated by the originator or the protocol has finished successfully. He then proceeds to compute the message.

The deadline time chosen must be less than the time that it will take the recipient to successfully compute the message. If the deadline chosen is greater than the time taken for the recipient to compute the message, then it will be very easy for the recipient to compute the message at each step and determine the last stage of the protocol execution. This makes it easy for him to obtain his non-repudiation evidence without sending the acknowledgement.

2.2 Probabilistic model checking

Probability helps to model uncertainties and performance (6) in the sense that it allows us to be able to quantify rate of failures (e.g. “the probability that the protocol will fail if both parties are honest is 0”), for quantitative analysis of

software and systems (e.g. “what is the maximum expected time for the protocol to terminate?”) or statistical analysis of behaviours. Probabilistic model checking involves a **probabilistic model construction** and **implementing probabilistic model checking algorithms** (6) using a probabilistic model checking tool e.g. PRISM (4) (5) to validate the results. Types of behaviours that can be modelled include: *probabilistic*, *non-deterministic*, *game* and *real-time behaviours*.

Probabilistic model checking Models include:

- **Discrete –Time Markov Chains (DTMCs):** have probabilistic and discrete time behaviours.
- **Markov Decision Processes (MDPs):** have probabilistic, non-deterministic and discrete time behaviours.
- **Continuous –Time Markov Chains (CTMCs):** have continuous time and probabilistic behaviours.
- **Probabilistic Time Automata (PTAs):** have probabilistic, non-deterministic and real-time behaviours.
- **Game models:** a game model is similar to the MDP model, but differs due to the fact that it is modelled to consider interactive behaviours of two or more parties competing against each other with the possibility that one party might be trying to take advantage of the other party.

The PRISM model checker (4) (5) supports the probabilistic models MDP, DTMC, CTMC, PTAs and has an extension to support games (7). It is the model checking tool of choice for this project.

The DTMC model will be the model need in this project and is further discussed in Section 2.4 below.

2.3 PRISM model checker:

PRISM is a probabilistic verification tool for automatic verification of systems that exhibit stochastic behaviours (4) (5). A high level model of the system to be analysed is constructed. After which this model constructed and its properties to be checked are translated into the PRISM simple state based language. PRISM helps to give us quantitative results (either through graphs or figures) about the probabilistic model constructed. Experiments are carried out on the model and properties to automatically generate graphs. The graphs can then be further analysed. Also, verification of the properties, gives probabilistic figures of the expected results, which are also analysed.

2.3.1 The PRISM Language

The PRISM language is a state-based language based on reactive modules formalism (8) while the property specification language integrates various temporal logics e.g. probabilistic computation tree logic PCTL (9) (10), continuous stochastic logic CSL (11) (12), linear time logic LTL (13) (14) and extensions for costs/rewards specifications (4) (5).

Modules and *variables* are the fundamental components of the PRISM language. Just like any other common programming language (e.g. java) *variables* can be declared and instantiated. *Variables* could be either numbers or Booleans. A *model* in PRISM is made up of one or more *modules* incorporated and interacting together. The construct of a *module* in a model is of the form: **module** name ... **endmodule**. A *module* is usually given an identifying name and consists of a number of local *variables* and *commands*. The state of the module is dependent on the values of these variables at any given time. While a set of *commands* (of the form $\square \text{ guard} \rightarrow \text{prob_1} : (\text{update_1}) + \dots + \text{prob_n} : (\text{update_n});$) is used to describe the behaviour of each *module* (4) (5). The square brackets \square means the start of a new command. The **guard** is a declaration over all variables in the model and if the guard is true, transitions will be made based on the updates and probability. An **update** describes the new value a variable should be in a module and **prob** is the probability with which the update should occur. If no probability is assigned to an update, then it is assumed that all the updates occur with equal probability which must add up to 1. A command can be labelled with an action at the start of the command e.g. **[req]** for sending or receiving a request. Such an action can be used to synchronize the transitions of different modules (5). Note that by default, PRISM allows modules to synchronize over all their common actions.

A model can also have rewards incorporated in it; to give additional information about the system. Reward structures in a model have a similar construct as that of the *module* i.e. **rewards** “label” ... **endrewards**. Just like modules have identifying names; rewards can have labels (if there are more than one reward structures, this helps to identify the reward being referred to). Rewards can be assigned to both state and transitions of a model. State rewards are usually of the form **guard : reward**. The **guard** is a predicate over all variables in the model and if the predicate is satisfied in a state then the related **reward** is assigned (e.g. **ack=n : 1** would assign a constant 1 to any state that **ack** is equal to **n**). The **reward** is an expression that can contain any variable in the model, a fraction of the variables, constants, etc. Transition rewards are usually of the form **[action] guard : reward**; which means that **action** labelled transitions in a state that satisfy the **guard** is assigned the corresponding **reward** (8). Example of a transition reward is **[req] true : 1** which means that if the action of sending or receiving a request is true in a state, then a reward of 1 is associated with that transition.

The PRISM language supports specification and analysis of properties based on *probability* of an event occurring and on *rewards* or *costs*. A discussion on the PRISM language used for representing properties is carried out in subsection 2.4.1 below.

The model written in PRISM language is loaded into PRISM, as well as its corresponding properties. Once loading of the model and its properties is done in PRISM, further progressions can be made to verify and analyse these properties.

2.4 Discrete Time Markov Chains (DTMCs)

DTMCs have probabilistic and discrete time behaviours. They are probabilistic state-transition systems where the probabilities of every state-transition in the system must add up to 1. In a DTMC, there is at least one outgoing transition in

every state and every state is independent of each other. Considering a set of random variables $\{X(k) \mid k=0,1,2,\dots\}$ where $X(k)$ is a state of the system being modelled at time-step k , a DTMC can be defined as a family of this set (15). An execution of the system modelled e.g. the protocol represents a **path** in a DTMC. In a DTMC model, time is modelled either as discrete time-steps or no information is assumed about the time transitions. However, discrete time-step is sufficient enough to model time accurately in certain situations (16) (5) and this is what is used to measure time in this project.

A DTMC is a tuple (S, s_0, P, L) where:

- S is a set of states
- s_0 is the initial state of S
- $P : S \times S \rightarrow [0,1]$ is the transition probability matrix which is a stochastic matrix
Where $\sum_{s' \in S} P(s, s') = 1$ for all state of the set of states
- $L : S \rightarrow 2^{AP}$ is a state labelling function taken from a set of Atomic Propositions (AP). (17)

Figure 2 below is an example of a DTMC modelling a communication protocol.

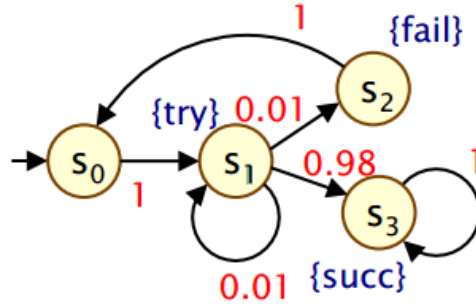


Figure 2: A simple DTMC model example (15) (5) (17).

From the example in figure 2 above we have a DTMC with:

- S set of states $\{s_0, s_1, s_2, s_3\}$ with s_0 being the initial state
- $AP = \{\text{try}, \text{fail}, \text{succ}\}$
- $L(s_0) = \emptyset$, $L(s_1) = \{\text{try}\}$, $L(s_2) = \{\text{fail}\}$, $L(s_3) = \{\text{succ}\}$

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In the example above, at an initial state s_0 the process starts, moves to a try state s_1 and tries to send a message, it succeeds with sending the message with a probability 0.98 and moves to success state s_3 ; with a probability 0.01 it fails to send a message and moves to fail state s_2 ; with a probability 0.01 the channel is unavailable so the process goes back to the try state s_1 where it tries to send a message again; from the fail state s_2 the process returns to the initial state s_0 to start all over again.

2.4.1 Property specification for a DTMC

DTMCs properties are articulated in PCTL (17). In PRISM, a **P** operator is used as a property specification to analyse the probability of an event occurring (5). When analysing the long-run (steady-state) behaviour of a model an **S** operator is used. Just like the **P** and **S** operators, an **R** operator is used in PRISM as a property specification for reward based properties. The **P**, **S** and **R** operators are allowed to take the following form '**P=?**', '**S=?**' and '**R=?**'. Properties of this form would return a numerical value rather than a Boolean (17).

DTMCs could have, *path based property* e.g. "what is the probability of unfairness of the protocol?", *a transient property* e.g. "what is the probability of being in an unfair state after k steps?", or an *expectation property* e.g. "what is the expected time-steps for the protocol to run?".

A wide range of path properties to be used with the **P** operator is supported by PRISM. Examples of path properties that can be used with a **P** operator are:

- **F**: "eventually" referred sometimes as "future" e.g. **P=? [F o=3]** returns a value for a path and gives the probability 'if a variable **o** is equal to 3 eventually at some point along the path'.
- **X**: "Next"
- **U**: "until" e.g. **P=? [ack<n U ack=n]** returns a value in a path and gives the probability 'if a variable **ack** remains less than **n** until eventually it is equal to **n**' (8).
- There are also the LTL-style properties in PRISM which provides more path properties for use with the **P** operator by combining temporal operators e.g. **P=? [F ("request" & (X "ack"))]** gives the probability of a request eventually being received and followed immediately by an acknowledgement (8).

Cost or rewards represents additional information about the system, rewards can accumulate either in a *state* or be acquired when an action is taken. Rewards in a DTMC are real-valued quantities assigned to states and/or transitions (16).

In PRISM there are four different types of reward properties that can be used with the **R** operator and they include:

- Reachability reward (**F** prop): reward that is accumulated along a path until a certain point is reached. For DTMCs, the reward is made up of state rewards along paths and transition rewards between states. E.g. of a reachability reward is **R=? [F o=3]** and gives the expected rewards accumulated along a path from the initial state until a state where **o** equals 3 is reached.
- Cumulative reward (**C<=t**): reward that is accumulated along a path until a given time bound **t** has passed. For DTMCs, the time bound **t** must be an integer and corresponds to steps. E.g. if we assign a reward of 1 to each transition of the model when a request is lost, **R=? [C<=k]** property would give the expected number of request lost within **k** time-steps of operation.

- Instantaneous reward (**I=t**): reward of a model at an instant time. It associates a path with the reward in the state of the path when t time (i.e. time-steps in a DTMC) passes exactly. E.g. $R=? [I=k]$ returns the expected reward at k time-steps.
- Steady-state reward (**S**): reward in the long-run and is not related to paths (5) e.g. $R \leq 0.8 [S]$ is true if from the initial state of the model the long-run reward is less than 0.8 (8).

A PRISM model may have several reward structures in it and in such a case, it is expected that the structure the property refers to is specified after the **R** operator and in braces $\{ \}$ (5). E.g. $R \{ \text{"unfair_o"} \}=? [F \text{ orig=done }]$ gives the rewards accumulated when the originator is in an unfair state until he reaches a done state.

2.5 Related Works:

I have focused my prior research work on papers that use probabilistic model checking for the analysis of the protocol proposed in (1).

However, it is worthy to note that games have been successfully used in the modelling and analysis of (non-probabilistic) non-repudiation protocols (18). In their approach alternating transition systems and alternating time temporal logic are used to model only the possible actions in the course of the protocol execution without sticking to a predefined order. They consider each party in the communication and the channel of communication as a different player. Furthermore, they formally express the properties of the non-repudiation protocol as *strategies*. And the protocol uses a strategy of trying to protect an honest party from every possible strategy of a dishonest/malicious party. But my own approach for the verification of the non-repudiation using games is based on PRISM-games (7) where probabilistic & non-deterministic behaviours of the non-repudiation protocol are exhibited in real time. The two parties (i.e. the originator and the recipient) are modelled as *players* and there are a finite set of states (7) and at each state, one player makes a choice from a set of available probabilistic transitions.

There have also been works based on process algebra equivalences e.g. (4) (8). Note however, the models do not consider real-time (they restrict to discrete steps) and restrict to either only non-deterministic or only probabilistic behaviours.

Two versions of the protocol by Markowitch & Roggeman (1) are considered in (6) to analyse a DTMC model of the protocol. In the first version, the recipient is honest and sends acknowledgements to each message, while in the second the recipient is malicious (tries to decode the message without sending the acknowledgement) and after receiving each message from the originator it makes a probabilistic choice as to whether to send an acknowledgement or to try and decode the message. Note that if it successfully decodes the message then it has "won" since it has the NRO evidence while the originator does not have the NRR evidence. If the recipient fails to decode the messages (because not all the messages have been sent), it can still try and send an acknowledgement.

However, in the second case, since it takes time to decode the message, the acknowledgement may not arrive before the deadline elapses thus making the originator terminate the protocol. The model has only discrete-time behaviour included: the version of probabilistic timed automata considered does not allow non-determinism, essentially if there is a delay over some time interval, the actual delay is made through a uniform distribution over the integers within the interval.

The timing behaviour of the model consists of assigning the minimum and maximum time it takes for: the originator and recipient to send messages; the recipient to try and decode the messages from the originator; and the deadline that must be exceeded by the recipient before the originator terminates the protocol (send no more messages).

The probabilistic behaviour is concerned with assigning the parameter of the geometric distribution used by the originator to select \mathbf{n} (the number of messages). In addition, in the model where the recipient is corrupt one needs to assign the probability that after each message the recipient decides to try and decode the messages. The properties the authors consider are:

- The probability that the protocol terminates correctly (for the honest recipient in the first version)
- The probability the malicious recipient wins (i.e. obtains the NRO evidence while the originator does not obtain the NRR evidence) in the second version.

The fact that there is only discrete time and there is no non-determinism means the model and analysis is restricted. In particular, by not allowing non-determinism (essentially replaced by uniform distributions) the analysis returns what can be considered as the "average" (*an average case does not rule out all possible behaviours making it hard to discover security flaws or attacks*) rather than the best or worst case behaviour. This model of the non-repudiation protocol has since been used as a case study for parametric probabilistic model checking in (19).

The paper in (19) is a DTMC model similar to the one discussed above in (20), the only difference here is that the probabilities associated with transitions are parameterized. The parameter \mathbf{p} is for the originator to decide whether to send the last message function or not. While the parameter \mathbf{q} is used to represent the recipient's decision whether to decipher the message or to send the acknowledgement.

A similar model is also considered in (21). This paper also considers how the originator can set its parameters in order to meet certain fairness requirements. However, this requires that the originator knows how the recipient will behave (i.e. the probability the recipient will reply to messages).

They consider one version of the protocol with two parameters:

- P_{geo} the parameter of the geometric distribution used by the originator for choosing the number of messages to send. They assume that even though geometric distribution may lead to an infinite number of steps, the originator will fix a maximum number of rounds r_{max} .

- P_{ack} the probability that the recipient sends an acknowledgement for each message of the originator. The recipient might decrease this probability by a fixed amount at each step or not. Their results inferred that a higher value of P_{ack} will give a lower cheating probability for a fixed P_{geo} value.

The authors' results show that the cheating probability (C_p) i.e. the probability that the recipient receives the last message without sending an acknowledgement) is directly proportional to the success probability P_{geo} and C_p decreases with an increased number of rounds r_{max} . They state their opinions about the originator selecting a low P_{geo} and a higher number of rounds r_{max} increases the cost of communication due to the longer run time. And by this, it shows that the security of the protocol can be increased at the expense of decreasing the performance (i.e. increase in the run-time). The authors' consideration of a DTMC model with no non-determinism is similar to the model considered in (22). And therefore, differs from the approach taken in this project, in the sense that it includes neither real-time nor non-deterministic behaviours. Also, it is worth noting that although the authors stated in the paper that they were going to view the protocol in (1) as a two player game, this was not actually done.

The security and performance issue discussed in (23) has led me to investigate security versus performance trade-offs. Trade-offs tends to affect the Quality of Service (QoS) in communication networks and the QoS issues of using non-repudiation protocols are discussed in (24).

In this paper, the practicability of using probabilistic non-repudiation protocol in a Mobile Ad Hoc Network (MANET) is tested by analysing the QoS issues in a MANET when using this protocol. They point out that it is not possible to use a non-repudiation protocol that involves the intervention of a trusted third party (TTP) in a MANET because of the structure of the MANET (MANETS are structured mainly for use in communications that involve critical data transfer between two parties), and thus no TTP is required in a MANET to ensure that this critical data is securely transferred. Also nodes in a MANET have limited resources. This limited their consideration to the protocol without a TTP of (1) and also the focus of this project. We saw earlier that not requiring a TTP helps to avoid communication bottlenecks that slows down or reduces performance. It is however important to note that because these protocols are time bounded (need to occur within a short timeframe) and have a requirement that the computational ability of the parties involved need to be considered. They seem rather unrealistic to apply them in a MANET (24). The author's state that, the originator must not choose a value of n that is too large or too small. As was seen previously, the larger the value of n the less the possibility of failure, but the longer it takes for the protocol to fully execute. And this longer time causes computational overheads. However, if n is too small, then it will be easy for the recipient to guess the last stage and this means that the possibility of the protocol being unfair increases. Therefore it can be seen that the **size of n greatly influences performance**. The paper also points out that **network resources are being wasted**, due to the fact that the originator has to send several message functions. Another issue considered in the paper is the **fact that the protocol is required to time-out**. As discussed earlier, the time taken for the receiver to send an acknowledgement must not be too long, or else he can easily decode the message without the protocol terminating and refuse to send the acknowledgement afterwards. Also the time must not be too short either so

that the recipient can have enough time to send the acknowledgement without the protocol terminating. These issues affect QoS of the protocol in a MANET. Also, there is the issue of ***delayed packets***. The authors point out that in a MANET, the routes for an acknowledgement to be sent may vary at each step of the protocol's execution. And as a result, because a different route is been used, some acknowledgement packets may be delayed, causing the originator to think that the recipient is trying to cheat and stopping the protocol.

Chapter 3 Requirements for Model Design

This chapter focuses on details of the design process, I state what was required for designing the model and identify the developmental changes needed and how they affect the fairness of the protocol. An explanation of the design decisions and proposed alternatives for modelling the protocol is also provided.

The initial motivation for undergoing this project was to model and analyse the probabilistic, non-deterministic and real time behaviours of the protocol specified by Markowitch and Roggeman in (1). As well as using stochastic two player games to model the interaction between the two parties of the protocol. In the initial design stage, the protocol was found to be unfair to the originator whether both parties behaved honestly or not. And this problem of unfairness discovered had to be improved first before going ahead to model and analyse the behaviours of the protocol. The project would have become too complex if improving fairness was to be considered at the same time with modelling the behaviours of the protocol stated above and applying it to stochastic two player game.

Therefore, the focus of the project changed from modelling and analysing the behaviours of the protocol and applying it to stochastic two player games to improving fairness of the protocol.

3.1 Discussion of problem found with regards to fairness

According to the non-repudiation protocol without a trusted third party proposed in (1), fairness is an important requirement of a non-repudiation protocol. This fairness is to ensure that at each step of the protocol run, either both parties receive their non-repudiation evidence, or none of them has any valuable information about their evidence. Markowitch and Kremer also defined *strong fairness* of a non-repudiation protocol similarly as a case where either both parties get their evidence or neither gets any valuable information at the end of the protocol run. They went further to define *true fairness* of a non-repudiation protocol as a case when *strong fairness* is provided and the exchange of non-repudiation evidence is successful in (22). The definition of fairness or strong fairness is said to be true by the authors if both parties are acting honestly. In the non-repudiation protocol as proposed by (1), the behaviours of the parties involved in the protocol run were not put into consideration. Their claim is that at each step during the protocol run, except the last step no party has an advantage over the other. They also did not consider the fact that the computing power of the recipient might be great enough such that at each step he actually gets some valuable information that contributes to his expected evidence.

It has been seen through model checking of the protocol using PRISM that the protocol is not fair enough irrespective of the intentions of both parties to act honestly or not. I observe two problems that seem to contribute to the unfairness of the protocol (especially for the originator).

- The first being that the recipient always gets messages first i.e. his non-repudiation evidence.
- Secondly, at each step during the protocol run, the originator gets nothing towards his non-repudiation evidence until the final *ack* is received from the recipient during the last step of the protocol run. While the recipient on the other hand gets his non-repudiation evidence incrementally.
 - o i.e. the NRR evidence = $\text{Sign}_R(\text{ack}_n)$
 - o while the NRO evidence = $\{\text{NRO}_i \mid i = 1, \dots, n\}$, with $\text{NRO}_i = \text{Sign}_O(f_i(\text{message}), O, R, D)$.

This unfairness problem makes it easy for the recipient to try and cheat and repudiate by computing the message without sending any evidence to the originator (especially if the computational power of the recipient in deciphering the message or guessing the last step is high enough). The need to address these problems stated above is important to cover all bases for fairness when designing and analysing the protocol.

3.2 Intended improvements of fairness

In the protocol as proposed in (1) the recipient always initiates the protocol by sending a request to the originator thus allowing the recipient to receive his evidence first. This puts the recipient at an advantage over the originator. A change in the ordering of the messages so that the originator can receive an *ack* first before sending a message at some point during the protocol run will help to reduce this advantage the recipient has over the originator. I argue that this change is feasible because it reflects common e-commerce practices where a “payment before service” notion is popular. Moreover a similar approach was taken in order to improve the fairness of the probabilistic contract signing protocol of Even, Goldreich and Lempel (EGL protocol) in (25). In their approach (which is similar to my proposed approach for improving fairness), the authors investigated (using PRISM) how the EGL protocol can be made fairer by changing the message ordering of the EGL protocol (this actually helped in improving the fairness of the EGL protocol).

Another thing that will help is if the originator gets *acks* incrementally that contributes towards his non-repudiation evidence he has at the end of the protocol run (i.e. NRR becomes a combination of all *acks* received), and not just his NRR evidence consisting of the final *ack* he receives at the last step. By so doing, both parties then get items that contribute to their expected evidences incrementally and not just the recipient. The expected evidence if this change is effected becomes:

- NRR evidence for the originator = $\{\text{Sign}_R(\text{ack}_i \mid i = 1, \dots, n)\}$
- NRO evidence = $\{\text{NRO}_i \mid i = 1, \dots, n\}$, with $\text{NRO}_i = \text{Sign}_O(f_i(\text{message}), O, R, D)$.

This would allow the originator to possess some evidence at each step of the protocol run like the recipient. This evidence can be presented to a judge in case of a dispute when the recipient tries to cheat the originator. Since the acknowledgement is signed by the recipient and time stamped in relation to received message, then in a case when the recipient tries to cheat/repudiate, the originator can present this evidence and the last message he sent after that, after this the judge checks the evidence and can conclude that the recipient in deed took part in the communication. This structure of accumulating evidences for both parties reduces the possibility of a denial on the part of both parties.

3.3 Assumptions made

As discussed earlier, even when both parties are honest, the protocol is unfair for the originator and only fair to him at the end of the protocol run. Therefore, a simple scenario where both parties act honestly is considered.

The Non-repudiation protocol in (1) assumes that each message must convey a time stamp in order to protect the recipient from replay attacks (which occurs when the originator makes use of old *acks* from previous communication with the recipient). Since the change in the ordering of messages may lead to the originator receiving the last *ack* before sending the final message, we require that the new models must convey timestamps as well in order to avoid any replay attacks.

According to Markowitch and Roggeman, the deadline after which the protocol is stopped has to be determined. To do this, two solutions were provided in (1), the first being that a deadline known to both parties publicly is used at each step of the protocol run. The second solution is that an operational channel is assumed to be used between the two parties such that messages passed through it must be received in the sequence it is passed within a known and constant time interval. In the models presented, it is assumed that the second solution is used i.e. an operational channel exists between the recipient and originator.

According to the protocol proposed by Markowitch and Roggeman, the probability that the recipient tries to cheat by not sending the last *ack* and computing the message is dependent on the geometric distribution that the originator uses to select n . However, geometric distributions as well as uniform distributions both model some **discrete random variables**. The major difference between the two is that a discrete random variable X that follows a geometric distribution with parameter p (success probability), has a probability distribution $P(X=x) = (1-p)^{x-1} \cdot p$. Where $x = 1, 2, 3, \dots$ (i.e. number of outcomes is to infinity). While if X follows a uniform distribution, then the values of x is chosen between an interval of two parameters a and b (e.g 1 and 20) and its probability distribution is $P(X=x) = 1/(b-a)$. Where $x = 1, 2, 3, \dots, n$ (i.e. the number of outcomes n is known and each is equally probable to happen) (26). The interval between which a discrete uniform random variable is distributed can be used to represent the minimum and maximum number of steps n that the originator can choose. Doing

this simplifies the proposed model and thus I assume the use of a DTMC model that follows a uniform distribution to select the number of messages.

3.4 Description of Models

Two models of the protocol proposed in (1) are designed and analysed. The first to be considered is a **deterministic model** and the second a **probabilistic model**. These models are considered for simplicity and for explanation of the unfairness problems of the protocol.

The deterministic model, considers the simplest scenario where both parties behave honestly and follow the protocol accordingly till it terminates successfully. In the deterministic model, \mathbf{n} is fixed in advance (\mathbf{n} represents the number of steps that the protocol should have). Each step consists of the exchange of both the *message* and the *acks* respectively.

While in the probabilistic model of the protocol, a parameter \mathbf{K} is introduced and \mathbf{n} is chosen uniformly over the interval $1, \dots, \mathbf{K}$. This is similar to the specification of the protocol which states that the originator chooses \mathbf{n} secrets from a geometric distribution. But for simplicity a uniform distribution is considered instead.

Different versions of both the deterministic and probabilistic models considered where:

1. The recipient gets non-repudiation evidence incrementally while originator gets nothing towards his evidence until the final *ack* is received (basic model of the protocol). This is considered in next chapter (*Chapter 4*).
2. The originator and recipient both get their non-repudiation evidence incrementally. This is considered in *Chapter 5*.
3. Both parties get their non-repudiation evidence incrementally and the ordering of the message at the last step is changed so that the recipient sends the last *ack* before receiving the final message. In e-commerce this idea (termed ‘pay before service’) is commonplace where customers require a service from a service provider. This is considered in *Chapter 6*.
4. Both parties get their non-repudiation evidence incrementally and the ordering of the message is changed after the originator sends the first message and receives its corresponding acknowledgement (i.e. the message sending changes after the first step, so that the originator can get an *ack* first at each subsequent step before sending a message). This is considered in *Chapter 7*.
5. Both parties get their non-repudiation evidence incrementally and the ordering of the message is changed, so that each party sends an item first

at each step in an interchangeable manner. This interchangeable manner of sending messages continues until the \mathbf{n}^{th} step is reached. This version of the models is considered in *Chapter 8*.

Chapter 4 Modeling and Analyzing the Protocol (version 1)

This chapter focuses on the modelling and analysis of the protocol based on version 1 of both the deterministic and the probabilistic models designed. Since version 1 of both models listed in Section 3.4, represents the model as specified by Markowitch and Roggeman in (1). The aim of this chapter is therefore, to explain, verify and analyse the unfairness problem of the protocol identified in Section 3.1. Supplementary materials e.g. PRISM codes, documentation, detailed evaluation result (i.e. simulation of the models and graphs obtained) are submitted along with this dissertation.

To analyse and verify the properties of the protocol, both the originator and the recipient are represented as separate *modules* in the PRISM language model. These modules communicate with each other, representing the transmissions of *evidence* between the two parties in line with the protocol.

Simple state transitions of the originator in the protocol run:

- An initial state S_0 where the originator awaits a request from the recipient.
- A state S_1 where he goes to send a message to the recipient.
- A state S_2 where he goes to receive acknowledgement evidence from the recipient.
- A final state S_3 to represent the end of the protocol run.

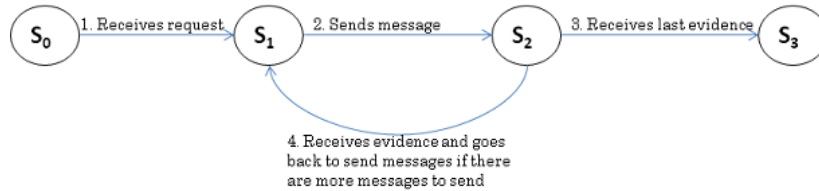


Figure 4a: State transitions of the Originator

Simple state transitions of the recipient in the protocol run:

- An initial state S_0 where he sends a request to the originator.
- A state S_1 where he receives a message from the originator.
- A state S_2 where he sends an acknowledgement for the message received from the originator.
- A final state S_3 when the protocol has ended.

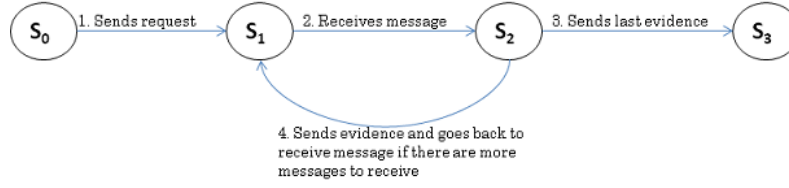


Figure 4b: State transitions of the recipient

4.1 The PRISM codes of the models explained

```

dtmc // model is a dtmc
const int n; // fix n in advance

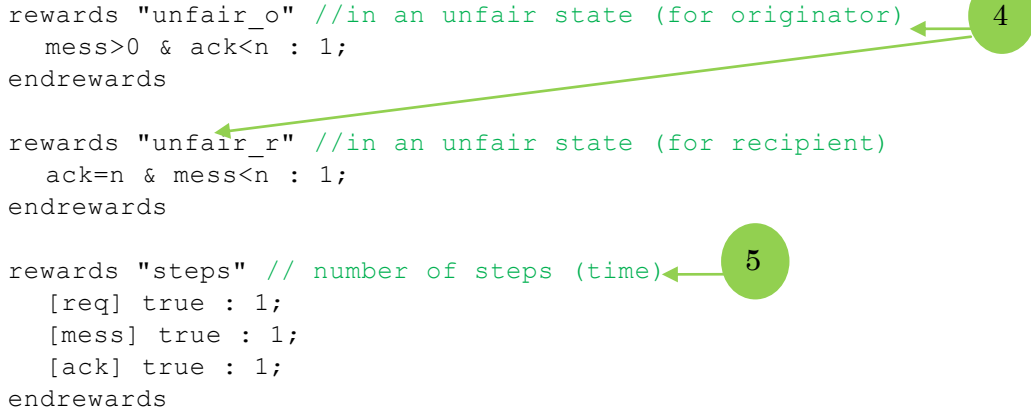
module originator
  o : [0..3]; // local state of originator
  // 0 initial state
  // 1 send messages
  // 2 receive acks
  // 3 done
  ack : [0..n]; // number of acks the originator has received
  [req] o=0 -> (o'=1); // gets request and proceeds to go send a
message
  [mess] o=1 -> (o'=2); // send message and goes to receive ack
  [ack] o=2 & ack<n-1 -> (o'=1) & (ack'=min(ack+1,n)); // receives
ack & not last; goes to send a message
  [ack] o=2 & ack=n-1 -> (o'=3) & (ack'=min(ack+1,n)); // receives
last ack and finishes
  [done] o=3 -> true; // finished so loop
endmodule

module recipient
  r : [0..3]; // local state of originator
  // 0 initial state
  // 1 receive messages
  // 2 send acks
  // 3 done
  mess : [0..n]; // number of messages the recipient has received
  [req] r=0 -> (r'=1); // sends a request and goes to receive
message
  [mess] r=1 -> (r'=2) & (mess'=min(mess+1,n)); // receives
message and proceeds to send ack
  [ack] r=2 & mess<n -> (r'=1); // sends ack & not last
  [ack] r=2 & mess=n -> (r'=3); // sends last ack
  [done] r=3 -> true; // finished so loop
endmodule

rewards "orig" // contribution to NRR the originator has
  ack=n : 1;
endrewards

rewards "recip" // contribution to NRO the recipient has
  true : mess/n;
endrewards

```



```

rewards "unfair_o" //in an unfair state (for originator)
  mess>0 & ack<n : 1;
endrewards

rewards "unfair_r" //in an unfair state (for recipient)
  ack=n & mess<n : 1;
endrewards

rewards "steps" // number of steps (time)
  [req] true : 1;
  [mess] true : 1;
  [ack] true : 1;
endrewards

```

Figure 4.1: Deterministic model version 1 PRISM code

From figure 4.1 above, the model type is specified first (i.e. it shows it is a dtmc model) and the variable **n** which is fixed in advance is declared as an integer. The variable **n** is not instantiated in the model, in order to allow us specify or set whatever integer value we desire **n** to be. Next in the PRISM code are the *modules* and *reward* structures used.

1. The module ‘**originator**’ is used to represent the originator and it shows the various states and actions the originator attains during the protocol run. A variable ‘**o**’ is declared and tells us the states the originator can acquire (**o** : [0...3] means the originator can either be in state 0, 1, 2 or 3). Variables such as the number of **acks** originator receives is also specified. Actions are represented by a set of guarded commands e.g. [**req**] represents the action of the originator receiving a request from the recipient (which is usually the start of the protocol run) and likewise [**mess**] represents the action of sending/receiving a message.
2. The module ‘**recipient**’ is used to represent the recipient and various states and actions that the recipient attains during the protocol run just like the originator module does.
3. In PRISM, rewards “**orig**” and “**recip**” are used to represent the NRO and NRR evidences gained by each party with time as proposed in (1). Where “**orig**” is a label used to identify the originator’s reward and “**recip**” the recipient’s reward.

Note: according to the specification of the protocol, the NRO and NRR evidence are as follows:

$$\text{NRO} = \{\text{NRO}_i \mid i = 1, \dots, n\}, \text{ with } \text{NRO}_i = \text{Sign}_O(f_i(\text{message}), O, R, D)$$

$$\text{NRR} = \text{Sign}_R(\text{ack}_n).$$

This means that the originator gets his non-repudiation evidence only at the last step i.e. when **ack=n**. This is achieved by assigning a real-valued quantity of 1 to the originator when he reaches a state where **ack=n**.

The recipient on the other hand gets contributions to his non-repudiation evidence incrementally when he receives a message (**mess**). And at exactly any instance that the recipient receives a message, a real-valued

quantity of the fraction of messages received to number of steps **n** (i.e. **mess/n**) is assigned to the recipient.

4. The rewards “**unfair_o**” and “**unfair_r**” are used to represent the advantage each party has during the protocol run. “**unfair_o**” is a label used to identify when the originator is in an unfair state (i.e. each time **mess>0 & ack<n**). When this happens, a cost of 1 is assigned to the originator. Likewise, “**unfair_r**” is a label used to identify when the recipient is in an unfair state (i.e. each time **ack=n & mess<n**) and when this happens a cost of 1 is assigned to the recipient. From this reward sturture, it is easy to show that the recipient has an advantage. This is due to the fact that there is actually no time where **ack** is equal to **n** and the message is less than **n**; since the recipient always sends a message first.
5. The reward “**steps**” is used to measure time used for the protocol run. This is achieved by allocating a reward of ‘1’ to each individual step of the protocol run. i.e. whenever the recipient and originator are either sending and receiving a request [req], sending or recieving a message [mess] or sending and recieving an acknowledgement [ack].

Figure 4.2 below shows the results obtained after parsing the model in PRISM and simulating it. In version 1 of the deterministic model as seen in Figure 4.2, it is observed that the reward for the recipient is accumulated incrementally until the last step, while the originator on the other hand attains his reward only after receiving the last ack. This means that the recipient always has an advantage over the originator at each step (making the protocol unfair to the originator). Through this, the protocol gives the recipient an opportunity at each step to cheat (especially if the computational power of the recipient is high enough). We can also observe that the protocol is never unfair to the recipient but unfair to the originator.

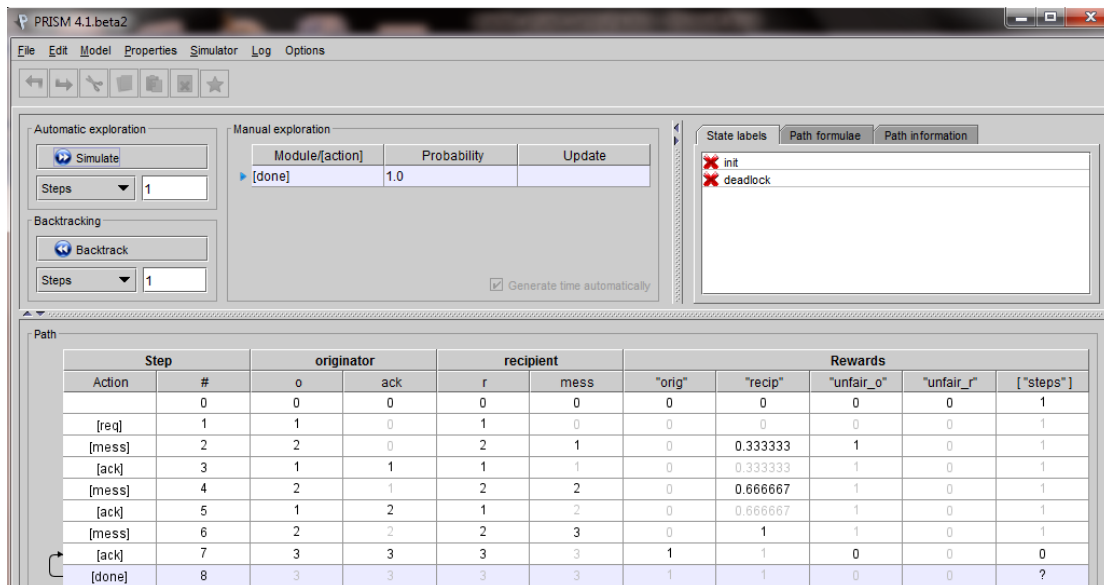


Figure 4.2: Simulation of version 1 of the deterministic model in PRISM when n=3

The PRISM code for version 1 of the probabilistic model (found in appendix A) is similar to that of the deterministic model, the major difference is that in the probabilistic model of the protocol considered, n number of steps for the protocol run is chosen randomly by the originator over $1, \dots, K$ where K is equal to 10 in the first version.

Simulation of the model in PRISM also gives a similar result to the simulation of the deterministic model and can be seen in appendix A.

After careful observation and analysis of the simulations, the next thing carried out is an analysis of the protocol's properties through the use of PRISM properties and rewards structure. The properties given in section 4.2 below are considered for the basic version of the models (i.e. version 1) and all other modified versions discussed in Chapters 5, 6, 7 and 8.

4.2 Properties and rewards considered

To analyse the fairness, viability and time-bound properties of the protocol, they were first written informally and then specified in the PRISM language (see Sections 2.3.1 and 2.4.1). The reward structure supported by PRISM was also used to analyse the properties. The first thing checked with respect to all models designed is 'if they terminate correctly'. The advantage the recipient has is demonstrated by analysing the fairness of the protocol in the different versions through an unfairness property. This unfairness property shows with "what probability one party reaches a state that the other party has obtained his non-repudiation evidence while the other has not?"

This property verified in PRISM as we shall see further, demonstrates that with a probability 1 the protocol is unfair to the originator and with a probability 0 it is unfair to the recipient (for a certain number of steps n chosen and whether both parties are behaving honestly or not).

A full list of the properties analysed is as follows:

1. **P=?[F o=3 & r=3]** ("what is the probability that the protocol terminates correctly?"). It reports the probability from the initial state of the model of the originator and recipient reaching a state 3 (done state in the model).
2. **R{"orig"}=? [I=k]** and **R{"recip"}=? [I=k]** ("What contributions towards their non-repudiation evidence do both parties have at k steps?"). Where **orig** refers to the originator, **recip** refers to the recipient and **[I=k]** is an instantaneous reward property that associates with a path the reward "**orig**" or "**recip**" in the state of that path when k steps have elapsed (5).

These properties analyses fairness from a different point of view and the viability property as stated in (1). It demonstrates that if both parties follow the protocol correctly, then at the end the exchange of NRO and NRR succeeds with a probability 1. However, it is important to note that

this successful exchange of evidence at the end of the protocol run doesn't change the fact that the recipient always has an advantage over the originator whether both parties are acting honestly or not.

3. $P=?[F(\text{ack}=\mathbf{n}) \ \& \ (\text{mess}<\mathbf{n})]$ and $P=?[F(\text{ack}<\mathbf{n}) \ \& \ (\text{mess}=\mathbf{n})]$ “how fair is the protocol to the originator? and how fair is it also to the recipient?” where \mathbf{n} is the number of steps of the protocol run, **ack** is the number of acknowledgements received and **mess** stands for the number of messages received. This property analyzes fairness by answering the question “What is the probability of reaching a state where one party has his expected evidence while the other does not?”. It reports the probability from the initial state of the model of reaching a state where acknowledgement received is equal to \mathbf{n} and the messages is less than \mathbf{n} and vice versa.
4. $R\{\text{"unfair_o"}\}=?[C\leq\mathbf{k}]$ and $R\{\text{"unfair_r"}\}=?[C\leq\mathbf{k}]$
 - a. Where “unfair_o” and “unfair_r” are rewards associated with a state of unfairness $(\text{ack}<\mathbf{n}) \ \& \ (\text{mess}=\mathbf{n})$ for the originator and $(\text{ack}=\mathbf{n}) \ \& \ (\text{mess}<\mathbf{n})$ for the recipient. $[C\leq\mathbf{k}]$ corresponds to the reward cumulated along a path until \mathbf{k} steps have elapsed (5).
 - b. The above properties are used to analyze the fairness property of the non-repudiation protocol as proposed in (1) in relation to time (\mathbf{k} steps). It addresses the question “what is the expected time spent in an unfair state after \mathbf{k} steps?” for both the originator and recipient.
5. $R\{\text{"unfair_o"}\}=?[F \ o=3 \ \& \ r=3]$ and $R\{\text{"unfair_r"}\}=?[F \ o=3 \ \& \ r=3]$ (These properties are similar to the above stated properties; the only difference is that this property demonstrates the total time spent in an unfair state by each party). Where $[F \ o=3 \ \& \ r=3]$ corresponds to the reward cumulated along a path until the protocol terminates (which is state 3 for each party in the PRISM code translation of the models).
6. $R\{\text{"steps"}\}=?[F \ o=3 \ \& \ r=3]$ “what is the required number of steps (time) for the protocol to terminate successfully?”.

4.3 Analysing the properties of the protocol in PRISM

4.3.1 Probability that the protocol terminates correctly

The probability that version 1 of both models terminates correctly is the same and is equal to 1. Whether \mathbf{n} varies from 1,...,20 in the deterministic model or \mathbf{n} is chosen randomly from 1... \mathbf{K} in the probabilistic model where \mathbf{K} varies from 1,...,20 (seen in figure 4.4 below).

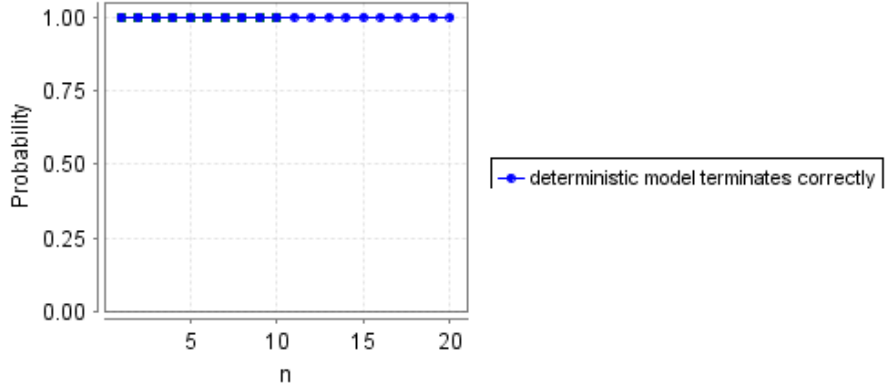


Figure 4.4: Probability that the deterministic model terminates correctly when n varies from 1...20

4.3.2 Expected evidence each party gains

Figure 4.5a and 4.5b below shows what the evidence each party has during the protocol run is and that the viability property holds (according to Markowitch and Roggeman) which states that if both parties follow the protocol properly to the end, then exchange of evidence for both parties succeeds with a probability equal to 1. In version 1 of both models, it is observed that the originator gets nothing towards his evidence until the last *ack* is received while the recipient's evidence incrementally increases. The probabilistic model's graph is different because in the probabilistic model, n is being chosen randomly from an interval 1... K (similar to the protocol specification that says n is chosen randomly from a geometric distribution). While in the deterministic one n is fixed and equals 20 in the experiment.

Note: for other values of n & K , the graph has the same pattern.

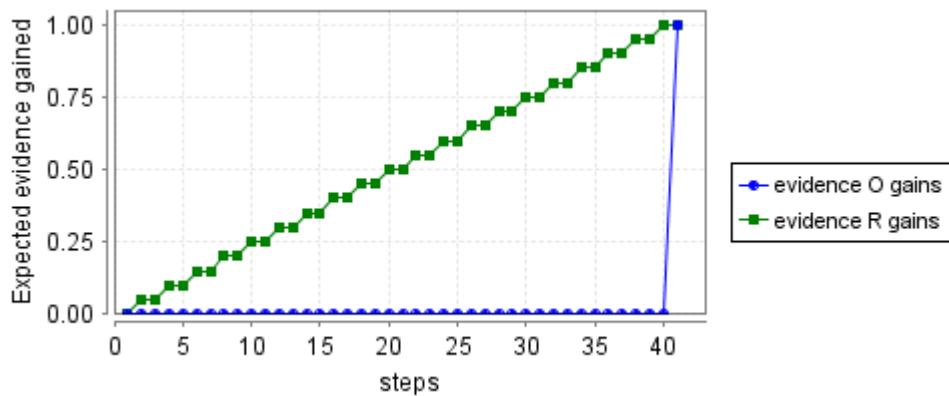


Figure 4.5a: Expected non-repudiation evidence each party gains after k steps when $n=20$ (for other values of n , the graphs have the same pattern).

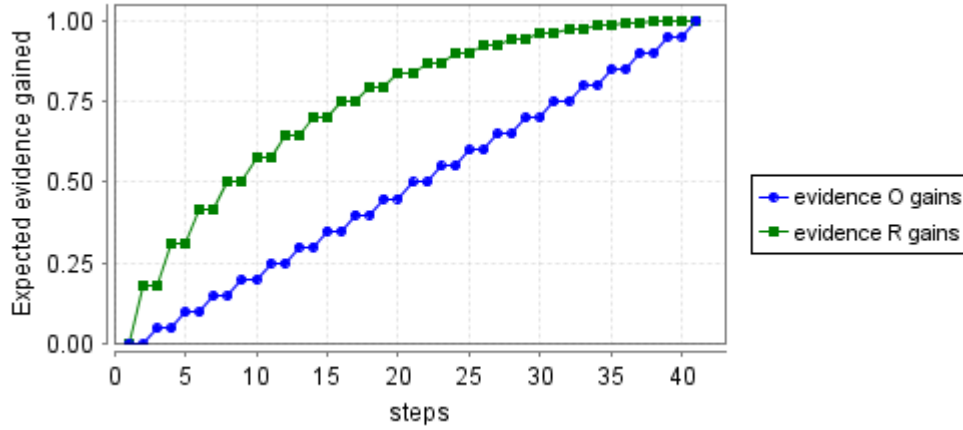


Figure 4.5b: Expected non-repudiation evidence each party gains after k steps when n chosen randomly from 1...20.

4.3.3 Probability of unfairness in the protocol

From figure 4.6 below, it is observed that the probability of the protocol been unfair to the recipient is 0 and probability of it been unfair to the originator is 1 in both models. This clearly verifies what has been stated earlier on, that the protocol is fair to the recipient with probability equal to 1 (unfair to the originator with this same probability).

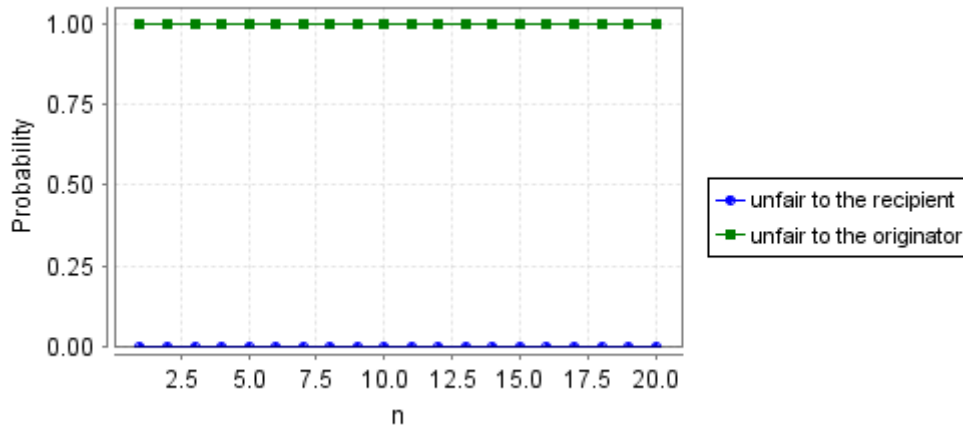


Figure 4.6: Probability of unfairness when n varies from 1,2,...,20 or if n is chosen randomly from 1...20

4.3.4 Time each party spent in an unfair state

From figure 4.7a and 4.7b below, it is observed that there is no time spent in an unfair state for the recipient while on the other hand, at each step after originator has sent a message he remains in an unfair state until he gets the corresponding acknowledgement. This goes to show that the protocol is unfair to O every time he sends a message to R and this is because R always receives a message first giving him an advantage where he can cheat by not sending the acknowledgement to O.

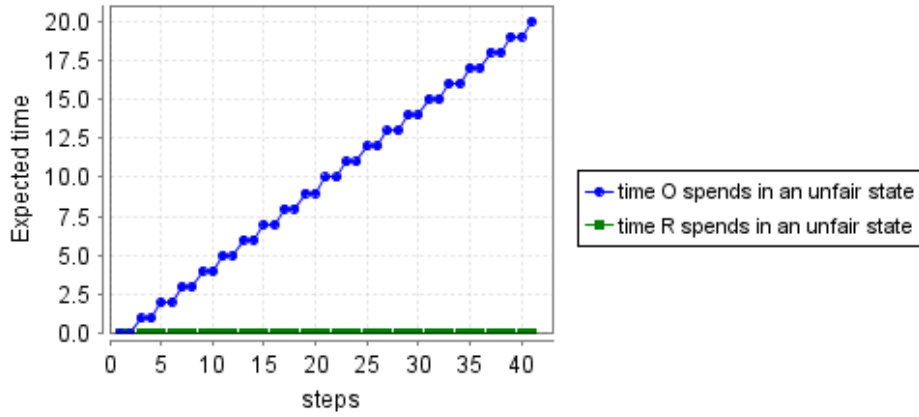


Figure 4.7a: Expected time spent in an unfair when n is fixed at 20 in the deterministic model.

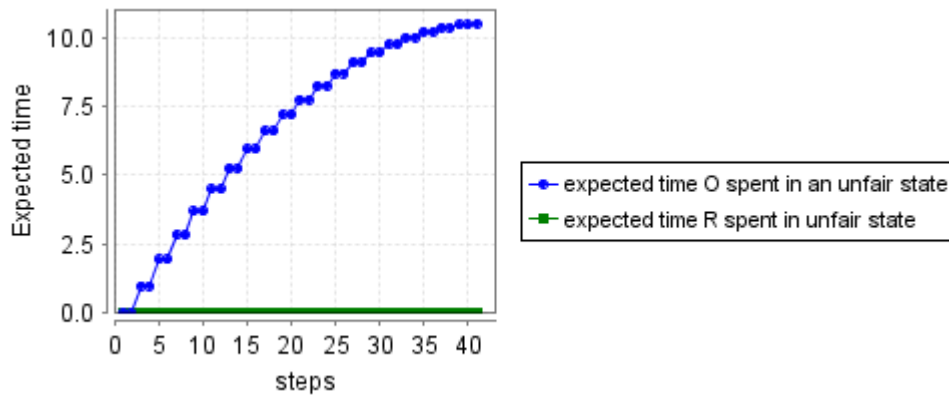


Figure 4.7b: Expected time spent in an unfair state when n is chosen randomly between 1,...,20 in the probabilistic model.

It is seen from figure 4.8a and 4.8b below that for any value of n & K , the originator spends at least some time in an unfair state while the recipient does not, the protocol is unfair to the originator and never unfair to the recipient. From both figure 4.8a and 4.8b below, it is observed that for a larger n value fixed in the deterministic model or K parameter set from which n is randomly chosen in the probabilistic case would lead to the originator spending more time in an unfair state.

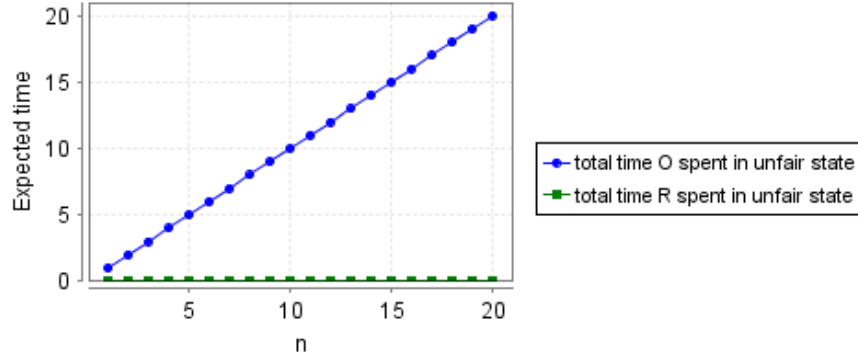


Figure 4.8a: Total time that both parties spent in an unfair state when n varies from 1,2,...20 in the deterministic model.

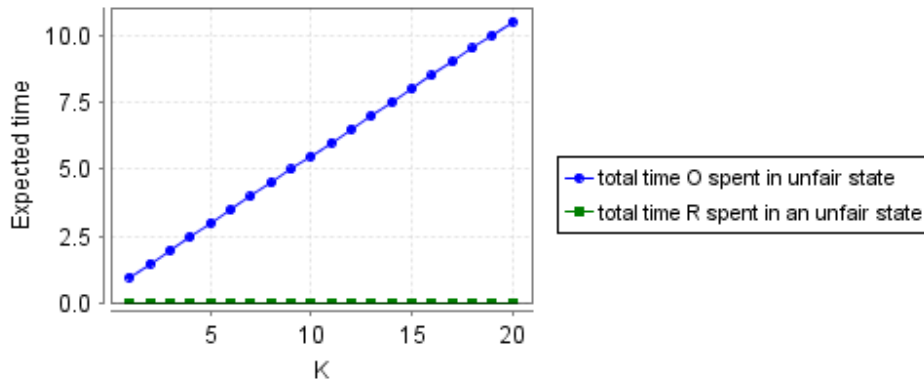


Figure 4.8b: Total time that both parties spent in an unfair state when K varies between 1...20 and n is chosen randomly from 1,... K in the probabilistic model.

4.3.5 Expected number of steps for the protocol run

The expected number of steps for the protocol to run is seen in figure 4.9a and 4.9b below, it is expected that if n is 20 (in the deterministic model) then 41 steps would be required for the protocol to run. Likewise, it would take 11 steps for the protocol to run if n is 5. In the probabilistic model however, the graph obtained is different because n is chosen randomly from 1,..., K intervals.

However, we can observe that the number of steps n chosen for the protocol run in both models is directly proportional to the time spent for the protocol to execute. I.e. a higher number of steps n equals higher number of messages transmitted and therefore more time spent in executing the protocol.

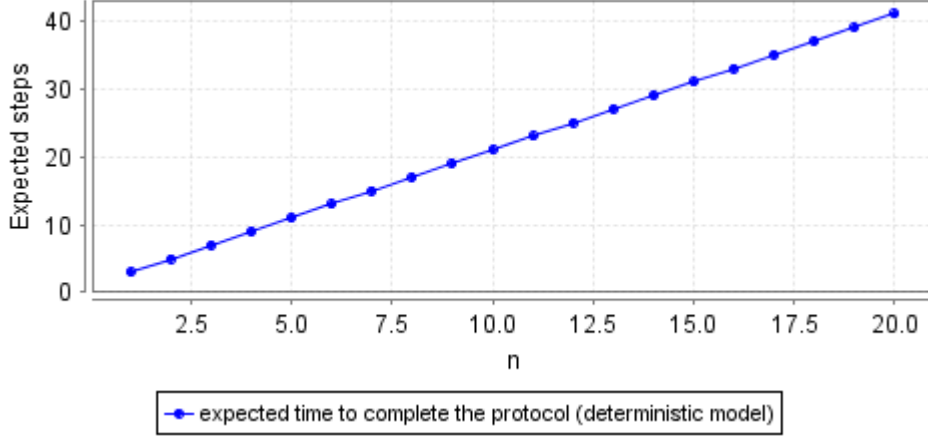


Figure 4.9a: Expected time to complete the protocol when n varies between 1,2,...,20 in the deterministic model.

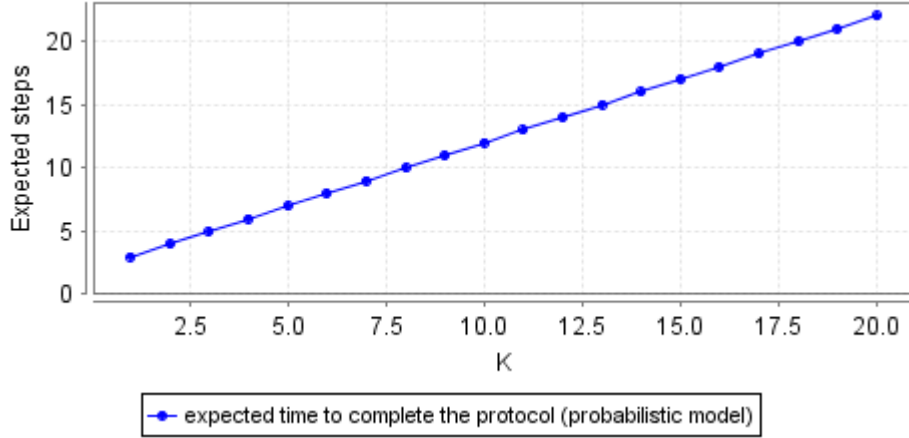


Figure 4.9b: Expected time to complete the protocol K varies between 1...20 and n is chosen randomly between 1,2,... K in the probabilistic model.

4.4 Conclusion

It has been observed through model checking of the protocol proposed by Markowitch and Roggeman in (1) that the protocol terminates correctly. If both parties behave correctly and finish the protocol to the end then they both get their expected evidence with a probability equal to 1, meaning the protocol is viable. However, it is observed that the recipient receives his evidence incrementally while the originator does not (he only receives his evidence when final *ack* is received), and therefore giving the recipient an advantage if he decides to cheat. This makes the probability of the protocol been unfair to the originator equal to 1 and the probability of it been unfair to the recipient equal to 0 during the protocol run (seen in figure 4.6 in subsection 4.3.3 above).

Chapter 5 Accumulating Evidences for both Parties (Version 2)

This chapter focuses on version 2 of the models, an introduction highlighting the difference between the basic models (version 1) discussed in Chapter 4 and this version is provided, property analyses of this new version in PRISM is given and a conclusion whether it helps to improve fairness or not is provided.

5.1 Introduction

The 2nd version of the models is when both the originator and recipient get their non-repudiation evidences incrementally. I.e. both the NRO and NRR become:

- $NRO = \{NRO_i \mid i = 1, \dots, n\}$, with $NRO_i = \text{Sign}_O(f_i \text{ (message), } O, R, D)$
- $NRR = \{\text{Sign}_R(\text{ack}_i \mid i = 1, \dots, n)\}$

This new NRR is a way of providing the originator with evidence at each step of the protocol run as earlier discussed. Since previously at each step except the last step, the originator has nothing to show as evidence if the recipient decides to cheat or repudiate.

In this second version, it is observed that if the $NRR = \{\text{Sign}_R(\text{ack}_i \mid i = 1, \dots, n)\}$ i.e. received at each step during the protocol run, this would allow the originator to possess some evidence at each step of the protocol run like the recipient. This evidence can be presented to a judge in case of a dispute, for example, when the recipient tries to cheat the originator. Since the acknowledgement is signed by the recipient and time stamped in relation to the received message, in a case when the recipient tries to cheat/repudiate, the originator can present this evidence and the last message he sent after that, after this the judge checks the evidence and can conclude that the recipient indeed took part in the communication. This structure of accumulating evidences for both parties reduces the possibility of a denial on the part of both parties. However, as we shall see; it does not change the fact that the recipient always has some advantage over the originator at each step of the protocol run.

The PRISM code for this version of the protocol can be found in Appendix A. It is similar to the codes for version 1 with the *modules* representing the originator and recipient remaining the same. The difference in the code for version 2 from version 1 is the change in some of the *rewards* structures used. E.g. the originator's reward representing his evidence is changed. To implement the changes, the fragment of codes used is shown in figure 5.1 below. From the fragment of code, we can observe that, instead of assigning a real valued quantity of 1 to the originator when he reaches a state that **ack=n**, a real valued fraction of *acks* received to the number of steps **n** (i.e. **ack/n**) is assigned to him at exactly

any instance he receives an acknowledgement. This is like having a trace property of the evidence that the originator has received at each step of the protocol run (i.e. instead of waiting till the last step to receive his evidence, the originator can now receive some evidence incrementally). Also, from figure 5.1 below, we can see that the rewards “**unfair_o**” and “**unfair_r**” changes and are used to represent the advantage one party has over the other at each step of the protocol run. This is another way of looking at fairness and “**unfair_o**” identifies when the originator is in an unfair state which happens each time his evidence received is less than the evidence the recipient has (i.e. each time **ack<mess**). When this happens, a cost of 1 is assigned to the originator. Likewise, “**unfair_r**” identifies when the recipient is in an unfair state (i.e. each time **mess<ack**) and when this happens a cost of 1 is assigned to the recipient. The reason for “**unfair_o**” and “**unfair_r**” rewards change in version 2 is because both parties are now receiving their evidences incrementally. And with this assignment of a cost at each step when a party has less than the other party, it is easy to show which party has an advantage at each step.

```
// this fragment of code replaces the reward labeled "orig" in
version 1 (seen in section 4.1)

rewards "orig" // contribution to NRR the originator has
  true : ack/n;
endrewards

// this fragment of code replaces the reward labeled "unfair_o"
and "unfair_r" in version 1 (seen in section 4.1)

rewards "unfair_o" //in an unfair state (for originator)
  ack<mess : 1;
endrewards

rewards "unfair_r" //in an unfair state (for recipient)
  mess<ack: 1;
endrewards
```

Figure 5.1: A fragment of the PRISM code for Version 2

With this new evidence accumulation for both parties, the protocol is still never unfair to the recipient but always unfair to the originator at each step (this can be seen in the simulation in figure 5.2 below). This unfairness is also observed when analyzing the fairness property for the protocol in Section 5.2.

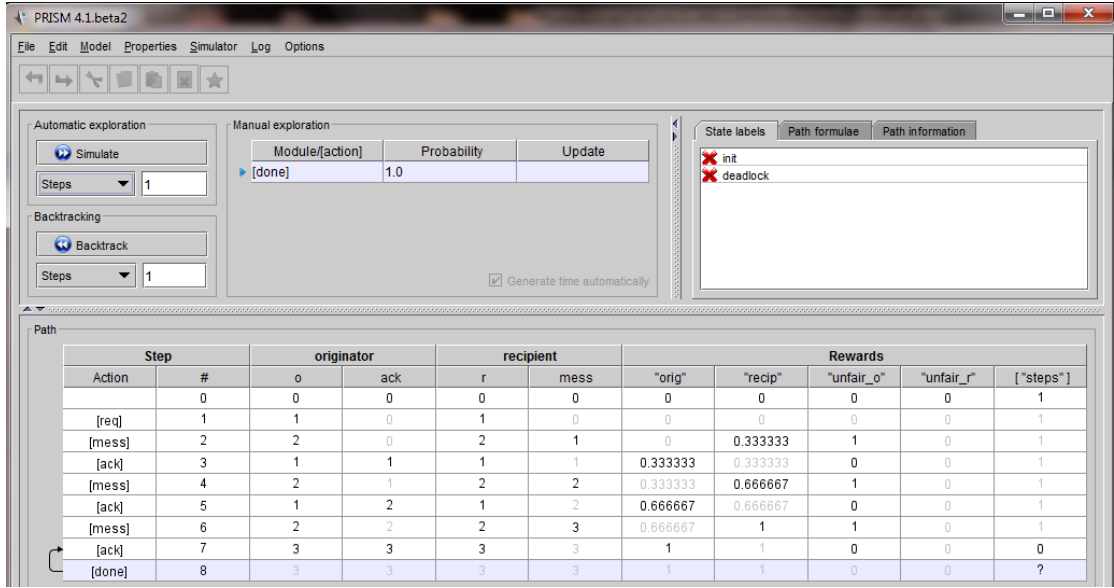


Figure 5.2: Simulation of version 2 of the deterministic model in PRISM when $n=3$

5.2 Analysis of Version 2

5.2.1 Probability of Version 2 terminating correctly

When analyzing version 2 of the models in PRISM, the probability of the protocol terminating correctly ($P=?[F \text{ } o=3 \ \& \ r=3]$) is equal to 1 which gives a similar graph to the one obtained when analyzing this property in subsection 4.3.1.

5.2.2 Expected evidence each party has after k steps in version 2

With the new change to the expected non-repudiation evidences that the originator gains, the graph to analyze the evidences gained changes in version 2 and can be seen in figure 5.3a and 5.3b below.

From figure 5.3a and 5.3b, it is observed that both parties now receive their evidences incrementally at each step and not just the recipient receiving his evidence incrementally like we saw in version 1 of the models (Section 4.3). Also, from the protocol specification, it is observed that this version is viable i.e. if both parties behave correctly then they both receive their expected item with a probability equal to 1. Notice that from these new graphs unlike the once obtained in chapter 4 subsection 4.3.2, the gap between the evidence both parties receive is closed. This is because they now both accumulate evidence together at each step.

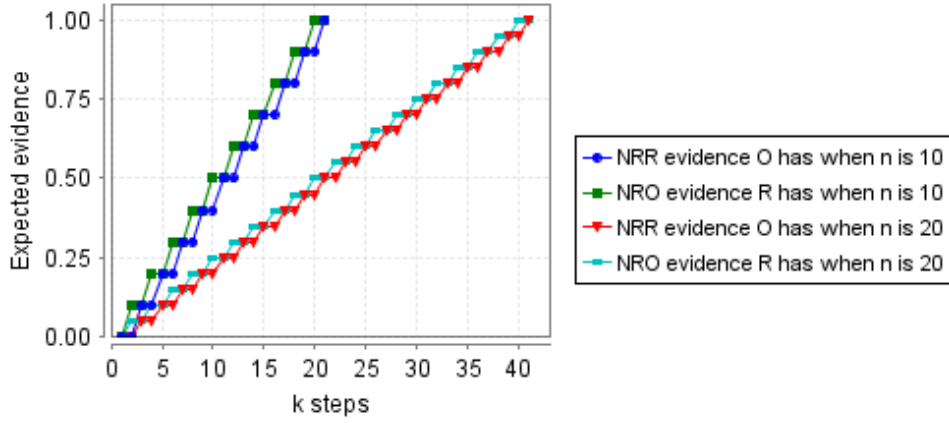


Figure 5.3 a: Expected non-repudiation evidence each party gains after k steps when n is 10 and 20 in the deterministic model.

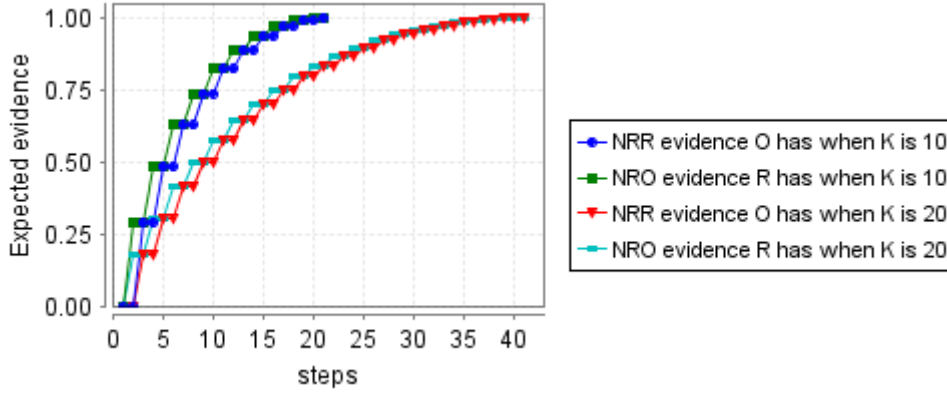


Figure 5.3b: Expected evidences gained by both parties when n is randomly chosen over $1, \dots, K$ intervals in the probabilistic model.

5.2.3 Probability of unfairness in version 2

The probability of unfairness in version 2 of the both models is the same as the probability of unfairness obtained in version 1 (subsection 4.3.3). It was observed that for n steps, the probability of the protocol been unfair to the recipient is 0 and probability of it been unfair to the originator is 1 in both models. This clearly verifies what has been stated earlier on irrespective of this change, that the protocol is fair to the recipient with a probability 1 (unfair to the originator with this same probability). However, with this new change, if the evidence accumulated by the originator is presented to a judge in a case of a dispute, it should be enough for him to conclude that the recipient indeed took part in the communication.

5.2.4 Time each party spends in an unfair state in version 2

Expected time that both parties spend in an unfair state after a certain number of steps and after the protocol has ended is the same as that observed in version 1 (see subsection 4.3.4).

5.2.5 Expected time (number of steps) to complete the protocol run in version 2

The expected number of steps for the protocol to end is same as in version 1 (subsection 4.3.5).

5.3 Conclusion

Version 2 of the models considers a scenario where both parties are receiving their non-repudiation evidences incrementally. This allows the originator to possess some evidence at each step like the recipient. In a case when the recipient tries to cheat/repudiate, the originator can present his evidence and the last message he sent to a judge. Since the evidence he received is signed by the recipient and time stamped, the judge can check the evidence and can conclude that the recipient in deed took part in the communication. The advantage of accumulating evidences for both parties is that it reduces the possibility of a denial on the part of both parties.

Through checking of the model properties in PRISM, I observed that this version terminates correctly as it should; it accumulates evidences for both parties as expected; it is viable (1); however, the protocol is still unfair with a probability equal to 1 for the originator and unfair with probability equal to 0 for the recipient. Lastly the number of steps n for the protocol run is directly proportional to the time spent for the protocol run.

Chapter 6 Evidence before Message at Last Step (Version 3)

This chapter focuses on version 3 of the models. An introduction highlighting the difference between version 2 discussed in Chapter 5 and this version is provided; property analyses of this new version in PRISM is given and a conclusion whether the changes in this new version helps to improve fairness or not is provided.

6.1 Introduction

Version 3 of the different models is when both parties are getting their non-repudiation evidences incrementally; and the message ordering at the last step is changed so that the recipient sends the last evidence first before receiving the final message. In e-commerce this can be termed as ‘pay before service’, which is mainly the case these days when customers require a service from a service provider. The difference in the PRISM code between this version and version 2 is that the actions performed by both parties in the *modules* are different. This is achieved by implementing new actions that allows the recipient send an *ack* first in the last step. The changes can be seen in the code fragment in figure 6.1 below.

// this code fragment replaces the originator module in the previous version

```
Module originator
  o : [0..3]; // local state of originator

  // 0 initial state
  // 1 send messages
  // 2 receive acks
  // 3 done

  ack : [0..n]; // number of acks the originator has received

  [req] o=0 -> (o'=1); // originator gets request and proceeds to
send message
  [mess] o=1 & ack<n -> (o'=2); // sends message and goes to
receive ack
  [mess] o=1 & ack=n -> (o'=3); // sends the last message and
finishes by going to the done state

  // receives ack & not approaching last step but has 1 ack so
he goes to send message.
  [ack] o=2 & ack<n-2 -> (o'=1) & (ack'=min(ack+1,n));
  // if he receives ack & entering last step, go to receive ack
again before sending last message
  [ack] o=2 & ack=n-2 -> (o'=2) & (ack'=min(ack+1,n));
  // if he receives last ack and n>1, go send message
  [ack] o=2 & ack=n-1 & n>1 -> (o'=1) & (ack'=min(ack+1,n));
  // if he receives last ack and n=1, he finishes
  [ack] o=2 & ack=n-1 & n=1 -> (o'=3) & (ack'=min(ack+1,n));
  [done] o=3 -> true; // finished so loop
```

```

Endmodule

// this code fragment replaces the recipient module in the
previous version

Module recipient
  r : [0..3]; // local state of originator

  // 0 initial state
  // 1 receive messages
  // 2 send acks
  // 3 done

  mess : [0..n]; // number of messages the originator has received
  counter : [0..1];

  [req] r=0 -> (r'=1); // sends a request and goes to receive
message
  // receive a message and goes to send ack
  [mess] r=1 & mess<n-1 -> (r'=2) & (mess'=min(mess+1,n));
  // if he receives last message and n=1, goes to send ack
  [mess] r=1 & mess=0 & n=1 -> (r'=2) & (mess'=min(mess+1,n));
  // if he receives last message and n>1, he finishes
  [mess] r=1 & mess=n-1 & n>1 -> (r'=3) & (mess'=min(mess+1,n));

  // send ack & not last but one, goes to receive message
  [ack] r=2 & mess<n-1 -> (r'=1);
  // send last ack and n=1, goes to done state
  [ack] r=2 & n=1 -> (r'=3);
  // send last but one ack and n>1
  [ack] r=2 & n>1 & mess<n-1 & counter=0 -> (r'=2) & (counter'=1);
  // send last ack and n>1 and counter is
  [ack] r=2 & n>1 & mess=n-1 & counter=1 -> (r'=1) & (counter'=0);
  [done] r=3 -> true; // finished so loop
Endmodule

```

Figure 6.1: A fragment of the PRISM code for version 3

From figure 6.1 above, the originator (i.e. the originator module) after receiving a request; sends a message as long as the evidence (**ack**) he has is less than **n** and goes to receive his evidence afterwards; if his evidence received is equal to **n** then he finishes and goes to the done state. When he is in a state of receiving his evidence i.e. if [**ack**] **o=2** ... there are several guards considered to determine what new state he will change to. E.g. “is he entering the last step (i.e. is **ack=n-2**)?”, if he is then he should go to state 2 to receive another evidence (i.e. $o' \rightarrow 2$); if he is not (i.e. **ack<n-2**) then he should go and send a message (i.e. $o' \rightarrow 1$). However, in a case where there are only two steps of the protocol run and extra guard to determine whether **n** is equal to 1 or greater than 1 is added.

On the other hand, in the recipient module (from figure 6.1), the recipient when in a state of receiving a message (i.e. [**mess**] **r=1**...) has some extra guards to consider before going to a new state. E.g. “if his messages received so far are less than expected (i.e. **mess<n-1**)” then he can go and send an evidence ($r' \rightarrow 2$). If there are only two steps for the protocol run, then an extra guard to determine this is added. In the recipient module, a **counter** variable is declared and added

as an extra guard to differentiate between two identical commands which can lead to the recipient having two different change states.

6.2 Analysis of Version 3

6.2.1 Probability of version 3 terminating correctly

In version 3, the protocol terminates correctly with a probability equal to 1 (just like the previous versions). This probability of the protocol terminating correctly is the same in both models.

6.2.2 Expected evidence each party has after k steps in version 3

The evidence that each party gains at each step remains the same as version 2 since both parties are getting their evidence incrementally, but the only difference is that at the last step of the protocol run, the originator gets the final ack before sending the last message. And it is expected that this will put the originator in an advantageous position since he obtains his full evidence first before the recipient gets his own complete evidence. However, if both parties act honestly, then they are both expected to receive their full expected evidence with probability equal to 1 (i.e. this version is viable and is seen in figure 6.2 below). The difference between this version and the previous versions as seen from the graph in figure 6.2 is that when n equal to 20 the originator receives his evidence first before the recipient does.

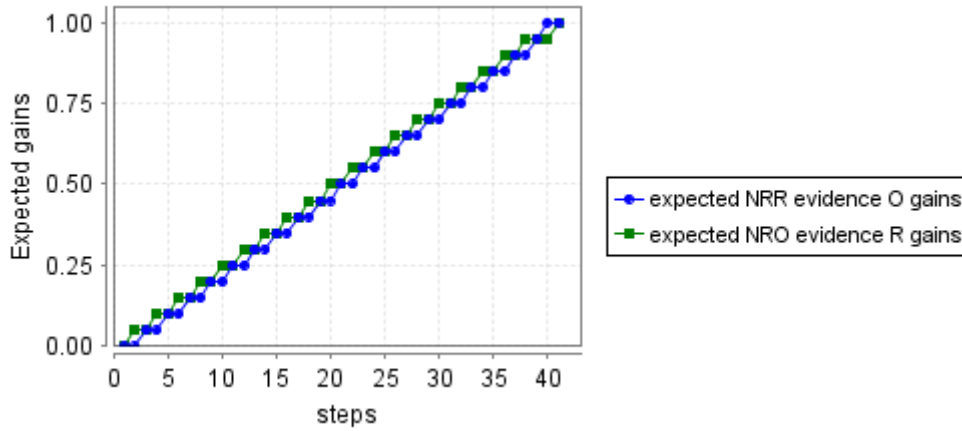


Figure 6.2a: Expected evidence gained by both parties at each step of the protocol run when $n=20$ (deterministic)

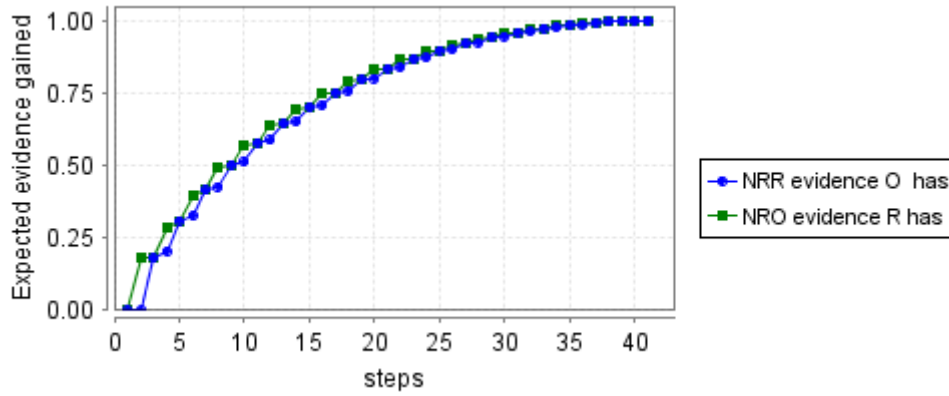


Figure 6.2b: Expected evidence gained by both parties at each step of the protocol run when n chosen randomly from 1,2,...20 (probabilistic)

6.2.3 Probability of unfairness in version 3

It is expected and is observed that when $n=1$ the protocol is unfair to the originator because he always has to start by sending a message first. But after this time, the unfairness is been switched from the originator to the recipient at the last step (seen in figure 6.3 below). Meaning, even with this change in the ordering of the message at the last step, the probability that the protocol remains unfair is still 1; but this time the unfairness changes to the recipient.

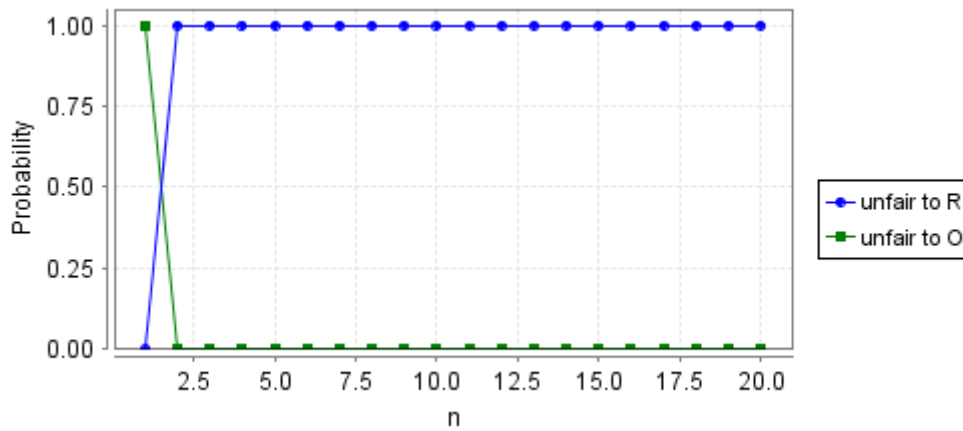


Figure 6.3a: Probability of unfairness for both parties when n varies from 1,2,...20 (deterministic model)

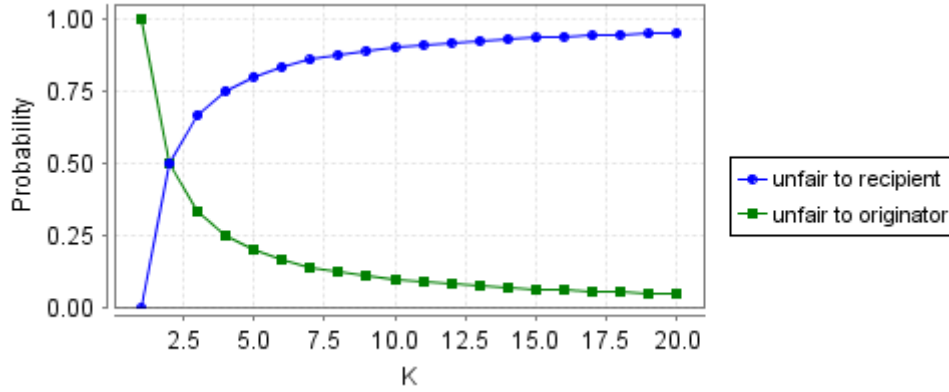


Figure 6.3b: Probability of unfairness for both parties when K varies from 1..20 and n randomly chosen from 1,...K (probabilistic model)

6.2.4 Time that each party spends in an unfair state in version 3

The time that each party spends in an unfair state is seen in figures 6.4 below. It is observed that for n steps of the protocol run the recipient now spends some time in an unfair state as well. This is due to the fact that in this version, the recipient has to send the evidence first before receiving a message from the originator at the last step.

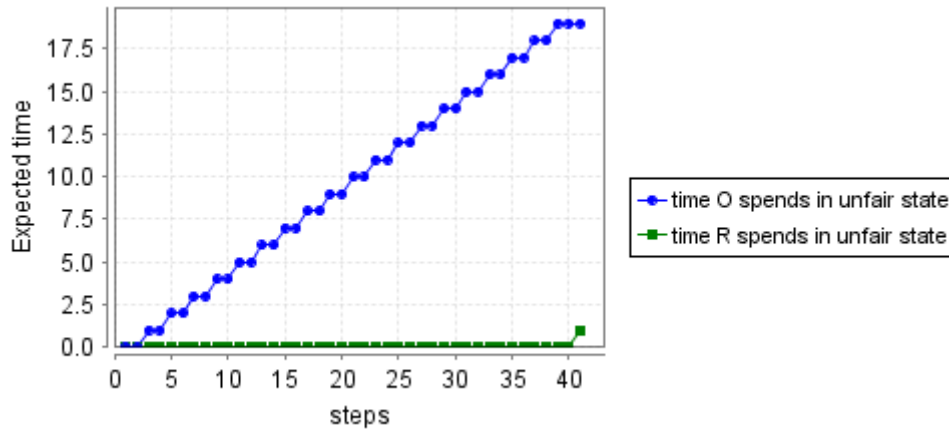


Figure 6.4a: Time both parties spend in an unfair state when $n=20$ (deterministic model)

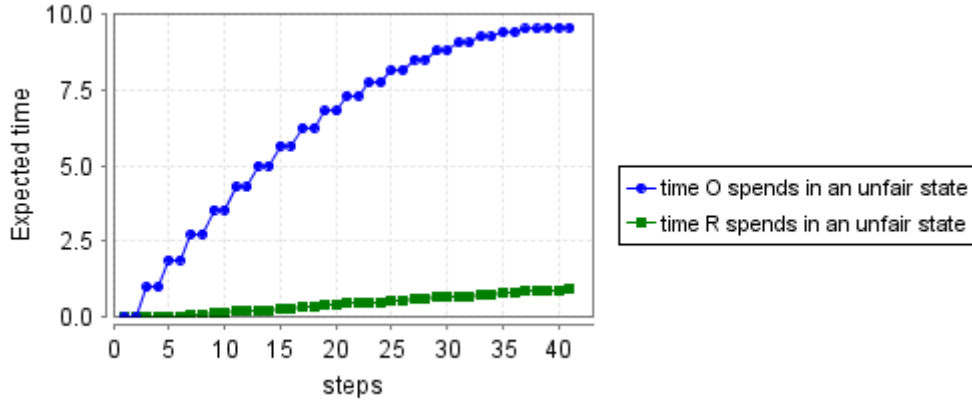


Figure 6.4b: Time both parties spend in an unfair state when $K=20$ (probabilistic model)

From figure 6.4a and b above, it is observed that O spends more time in an unfair state and this is because of the advantage that R has over O (i.e. O always sends a message first) in all subsequent steps except the last one. However, R now also spends some time in an unfair state unlike in the previous versions where R spends no time in an unfair state.

The total time that both party spent in an unfair state determined at the end of the protocol run is as follows:

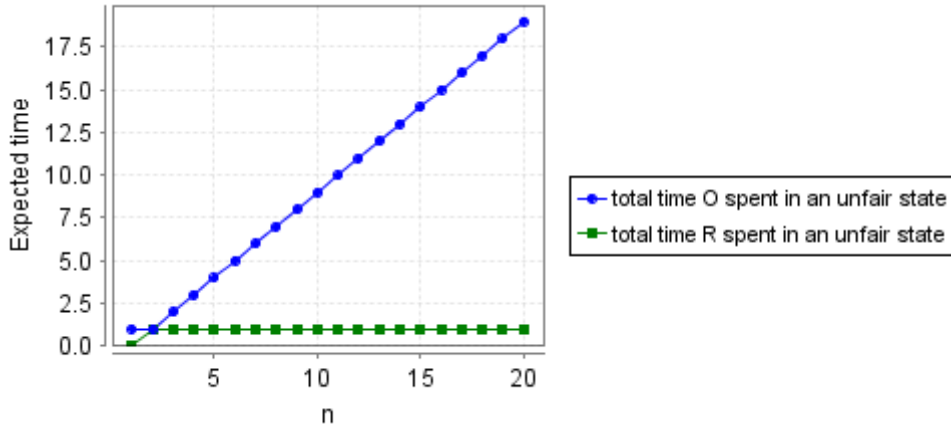


Figure 6.4c: Total time spent in an unfair state when n varies between 1...20 (deterministic model)

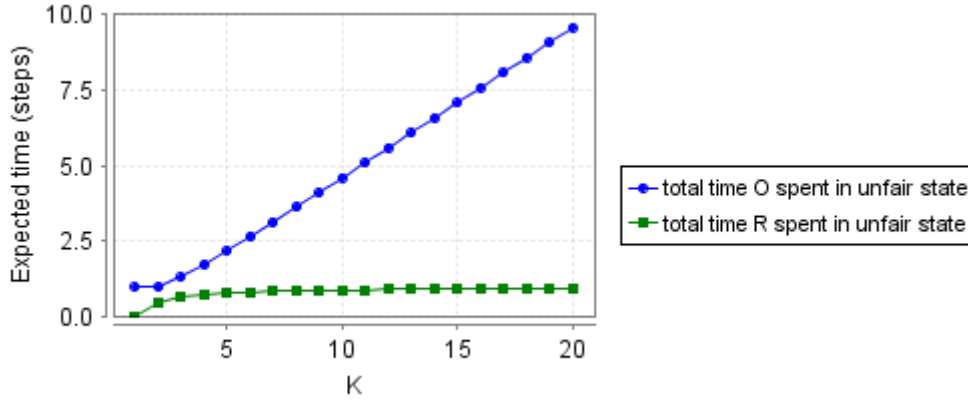


Figure 6.4d: Total time spent in an unfair state when $K=20$ and n is chosen randomly from $1...K$ (probabilistic model)

From figure 6.4c and 6.4d above, it is observed that as n increases, then the protocol returns to being unfair to the originator. But again, the recipient now spends at least some time in an unfair state.

6.2.5 Expected time (number of steps) to complete the protocol run in version 3

The expected number of steps to complete the protocol in version 3 of the models is the same as the number of steps that would be expected to complete the protocol in both version 1 and 2 of the models.

6.3 Conclusion

In version 3 analysed, it is observed that it did not to help in improving the fairness of the protocol. Rather the security of the protocol, which depends on the secret of n not being known by the recipient may most likely, be lost with this version of the protocol. This is as a result of the fact that for the recipient to know to send the final **ack** first before receiving a message at the last step he has to know that he is entering the last transmission. With the recipient knowing when the last step is, he can decide not to send the final **ack** but go ahead to compute the message (1). Also it is observed that if n increases, then the protocol returns to being unfair to the originator.

Chapter 7 Evidence before Message after Start (Version 4)

This chapter focuses on version 4 of the models, an introduction is provided highlighting the difference between versions 3 discussed in Chapter 6 and this version. The properties of the protocol are analyzed in this new version in PRISM and a conclusion is made whether the changes in this new version helps to improve fairness or not.

7.1 Introduction

Since version 3 (Chapter 6) tends to reduce the security of the protocol, a modified version 4 is considered in this chapter. The main difference between version 3 and version 4 is that the ordering of the message is changed after the first message and acknowledgment have been exchanged i.e. at the first step. This change would involve the recipient sending the *ack* evidence first at each subsequent step (after the first) before receiving a message. With this, the secret of *n* does not need to be revealed to the recipient. However, it is important to note that if *n* equals 2 then version 3 and 4 would be identical to each other. This is because after the first step, the second step in version 4 refers to the last step in version 3. The PRISM code to implement this change is seen in figure 7.1 below.

```
//in the originator module these codes replace the action code for
when the originator is receiving an acknowledgement

    // if he has received at least one ack & not last; he goes to
    send message.
[ack] o=2 & ack>1 -> (o'=1) & (ack'=min(ack+1,N));
    // receives last ack and n=1; he finishes
[ack] o=2 & ack=0 & n=1 -> (o'=3) & (ack'=min(ack+1,N));
    // receives first ack and n>1 (so now wait for ack before
    sending last message)
[ack] o=2 & ack=0 & n>1 -> (o'=2) & (ack'=min(ack+1,N));
    // receives last ack and n>1
[ack] o=2 & ack=1 -> (o'=1) & (ack'=min(ack+1,N));

// and this part replaces the code for sending acknowledgements in
the recipient module

    // sends ack & not last but one; goes to receive message
[ack] r=2 & mess>1 -> (r'=1);
    // if he sends last ack and has all his messages and n=1, he
    finishes
[ack] r=2 & mess=1 & n=1 -> (r'=3);
    // send last but one ack; and counter is 0 and n>1; he goes to
    send another ack
[ack] r=2 & mess=1 & counter=0 & n>1 -> (r'=2) & (counter'=1);
    // sends last but one ack; counter is 1 and n>1; goes to receive
    message
[ack] r=2 & mess=1 & counter=1 & n>1 -> (r'=1) & (counter'=0);
```

Figure 7.1 A fragment of the PRISM code for version 4

From figure 7.1 (comments included), if the originator is at the first step he receives an evidence after he must have sent a message first; if he has his first evidence already, he goes to receive another one; after which he sends a message; if he is to receive an evidence and has not, he goes to receive it; if he has sent all his messages and has received all his evidence; he finishes by going to the done state.

7.2 Analysis of Version 4 in PRISM

7.2.1 Probability of version 4 terminating correctly

In version 4, the probability that the protocol terminates correctly remains the same as that of version 3 (discussed in subsection 6.2.1 i.e. with probability equals to 1 the protocol terminates correctly in both models).

7.2.2 Expected evidence each party has after k steps in version 4

The expected evidence gained by each party at each step is shown in figure 7.2a and b below. From the figures, it is observed that at the first step of the protocol run, R gets his evidence first, but in subsequent steps O receives his evidence first. This is due to the fact that the message ordering has been changed so that after the protocol starts, the originator can receive a payment before sending a message. This version of the model is viable (1).

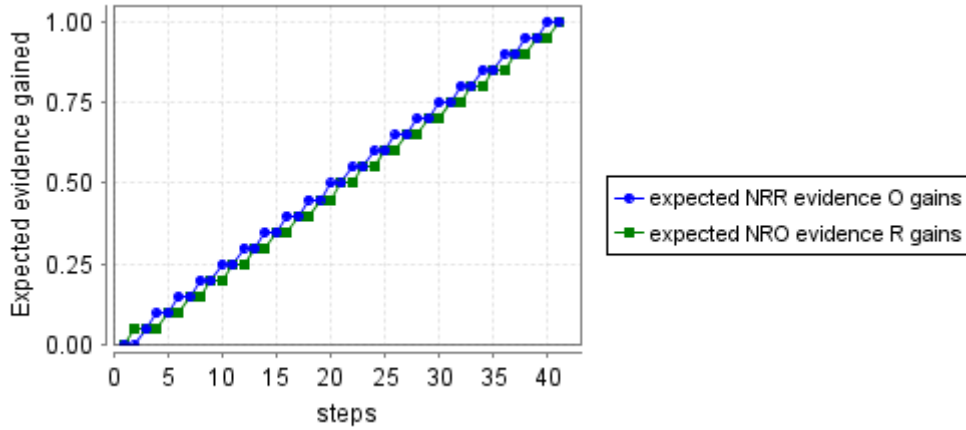


Figure 7.2a: Expected evidences gained when n=20 (deterministic)

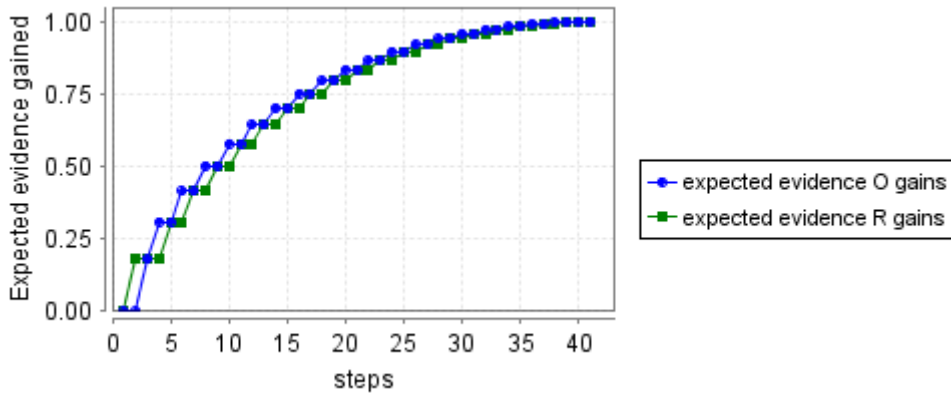


Figure 7.2b: Expected evidences gained when n is randomly chosen from interval 1...20 (probabilistic)

7.2.3 Probability of unfairness in version 4

The probability of unfairness measured with respect to version 4 remains the same as in version 3 (subsection 6.2.3). And it is observed that the protocol remains unfair to the originator when n equals 1 but this unfairness is transferred to the recipient at subsequent n steps. This happens as a result of the originator always having to send a message first to the recipient at the start of the protocol run, then at subsequent steps is when the change occurs.

7.2.4 Time each party spends in an unfair state in version 4

The time spent by each party in an unfair state is seen in figures 7.3a and 7.3b below.

From figures 7.3a and 7.3b it is observed that the originator spends some time in an unfair state; and will only more time in an unfair state than the recipient when n is equal to 1. On the other hand, the recipient tends to spend more of his time in an unfair state as compared to the originator. The difference in the time spent in an unfair state by both parties in this version as compared to version 3 (subsection 6.2.4) is that in this version it is the recipient who spends most of his time in an unfair state. This is due to the fact that after the first step, the recipient has to always make a payment first (i.e. provide the originator with some evidence) before he can receive any message from the originator. The security of the protocol is however maintained, since the recipient does not need to know the value of n because the message ordering changes at the start of the protocol run and if the recipient decides to guess n then the protocol will time out.

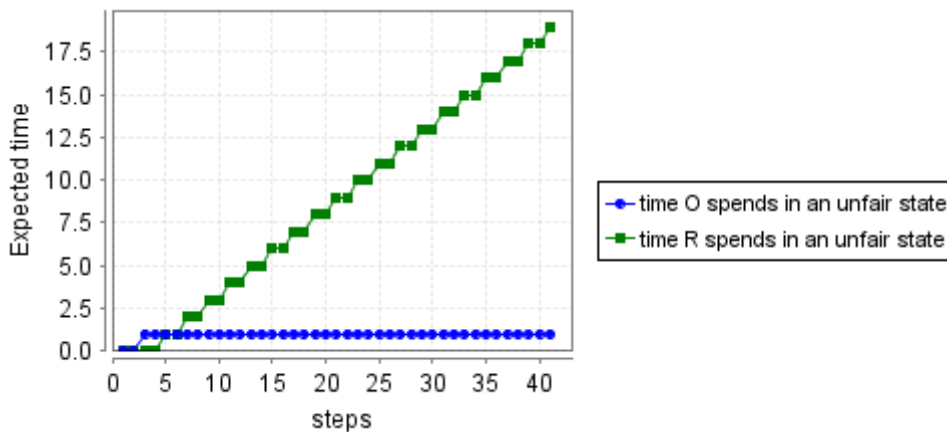


Figure 7.3a: Time spent in an unfair state after k steps when $n=20$ (deterministic model)

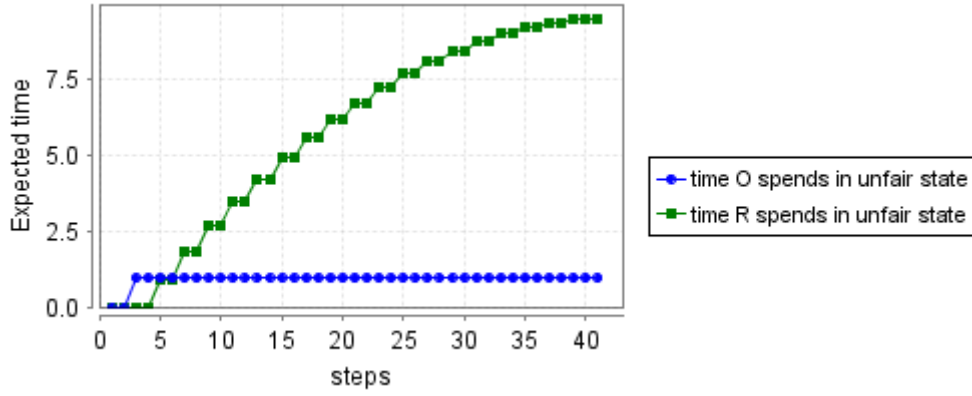


Figure 7.3b: Time spent in an unfair state after k steps when n is randomly chosen from 1...20 (probabilistic)

The total time each party spent in an unfair state in version 4 differs from that of version 3 in the sense that the recipient is the one who spends more time in an unfair state than the originator (seen in the figure 7.2c and 7.2d below).

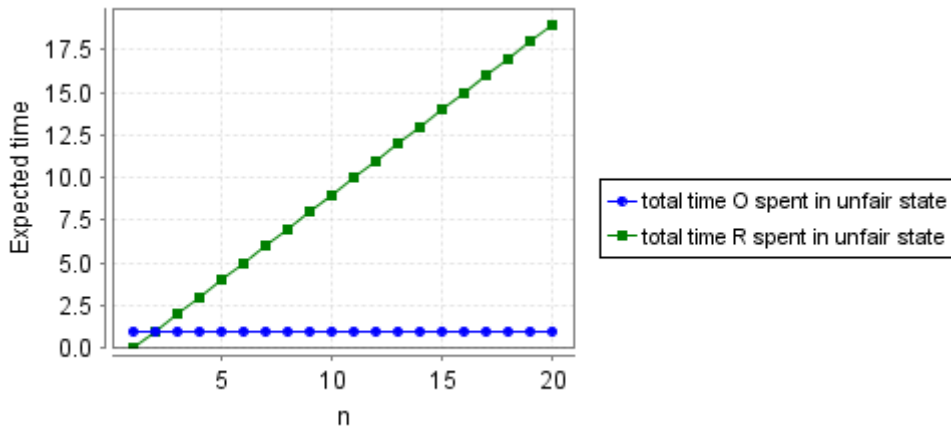


Figure 7.3c: Total time spent in an unfair state when n varies from 1...20 (deterministic)

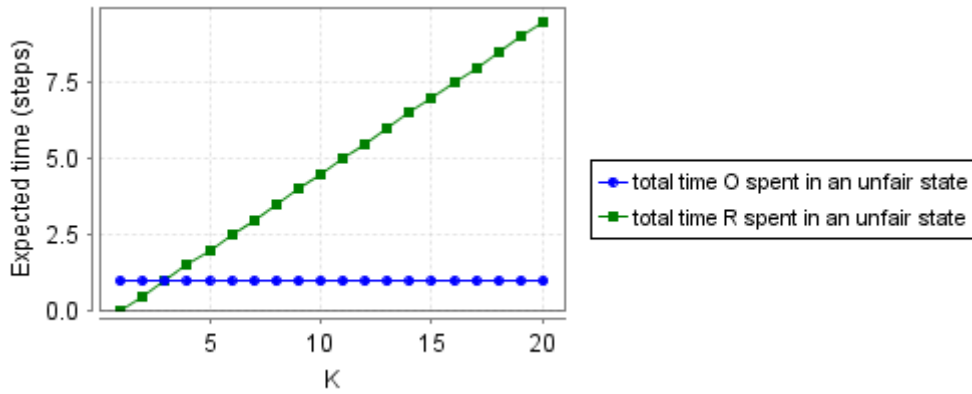


Figure 7.3d: Total time spent in an unfair state when K varies from 1...20 and n is randomly chosen from 1...K (probabilistic)

7.2.5 Expected time (number of steps) to complete the protocol run in version 4

The expected number of steps required for version 4 of the protocol to complete is the same as that of version 1, 2 and 3 of the model. From the graphs obtained, it is observed that the time for the protocol to run expressed as the number of steps is directly proportional to the value of n .

7.3 Conclusion

This version unlike version 3 preserves the security of n not being known to the recipient because the sending of an ack first before receiving a message is not done at the last step but immediately after the first step.

In version 4 of the model, the protocol will terminate correctly; both parties will receive evidences at each step and with a probability equal to 1 it is viable (1); the originator will spend only one step in an unfair state and this happens when $n=1$ while the recipient spends his time in an unfair state when $n>1$. This is due to the fact that at the start of the protocol run, when the originator receives a request from the recipient, he has to choose n number of steps for the protocol run, and then proceeds to send a message first to the recipient. Since the probability of unfairness is not improved in version 4 but only swapped from the originator to the recipient, then version 5 is considered in Chapter 8.

And lastly it is observed that a higher value of n chosen by the originator requires a higher time to complete the protocol (i.e. n is directly proportional to the time spent in the protocol run).

The PRISM codes and simulation of version 4 are found in Appendix A.

Chapter 8 Interchangeable Exchange (Version 5)

This chapter focuses on version 5 of the models, highlighting the difference between version 4 discussed in Chapter 7 and this version, property analyses of this new version in PRISM and whether the changes in this new version helps to improve fairness or not.

8.1 Introduction

In version 5, both parties get their *evidence* incrementally and the message ordering is changed, so that each party sends an item first at each step in an interchangeable manner as seen in figure 8.1 below.

```
R→O: recipient starts by sending a request.
O→R: originator after choosing n sends the first message.
R→O: recipient sends evidence.

R→O: in the 2nd step the recipient sends evidence first
O→R: the originator sends message after receiving evidence.

O→R: in the 3rd step the originator first sends a message.
R→O: the recipient sends evidence.

R→O: in the 4th step the recipient sends evidence first
O→R: the originator sends message after receiving evidence.

O→R: in the 5th step the originator first sends a message.
R→O: the recipient sends evidence.
.
.
.
.

R/O→O/R: in the nth sends evidence first
O/R→R/O: sends evidence after receiving what was sent earlier.
```

Figure 8.1 An illustration of the interchangeable way in which evidences are sent at each step in version 5

The PRISM code for this version can be found in appendix A, however a fragment of the PRISM code to implement this change in the way the messages are sent is shown in figure 8.2 and discussed below.

// The originator and recipient modules are replaced with the following codes

Module originator

```
o : [0..3]; // local state of originator
// 0 initial state
// 1 send messages
// 2 receive acks
// 3 done
ack : [0..N]; // number of acks the originator has received
```

```

o_even : [0..1]; // even or odd phase (starts odd)

    // get request (start in phase 1 which is odd)
[req] o=0 -> (o'=1) & (o_even'=0);
    // odd phase
[mess] o=1 & o_even=0 -> (o'=2); // sends a message
    // receive ack & not last
[ack] o=2 & o_even=0 & ack<N-1 -> (o'=2) & (ack'=min(ack+1,N)) &
(o_even'=1);
    // receive last ack
[ack] o=2 & o_even=0 & ack=N-1 -> (o'=3) & (ack'=min(ack+1,N));

    // even phase
[ack] o=2 & o_even=1 -> (o'=1) & (ack'=min(ack+1,N)); //receive ack
    // send message & not last
[mess] o=1 & o_even=1 & ack<N -> (o'=1) & (o_even'=0);
[mess] o=1 & o_even=1 & ack=N -> (o'=3); // send last message
[done] o=3 -> true; // finished so loop
Endmodule

Module recipient

    r : [0..3]; // local state of originator
    // 0 initial state
    // 1 receive messages
    // 2 send acks
    // 3 done
mess : [0..N]; // number of messages the originator has received
r_even : [0..1]; // even or odd phase (starts odd)

[req] r=0 -> (r'=1) & (r_even'=0); // send request
    // odd phase
    // receive message
[mess] r=1 & r_even=0 -> (r'=2) & (mess'=min(mess+1,N));
    // send ack & not last
[ack] r=2 & r_even=0 & mess<N -> (r'=2) & (r_even'=1);
[ack] r=2 & r_even=0 & mess=N -> (r'=3); // send last ack

    // even phase
[ack] r=2 & r_even=1 -> (r'=1); // send ack
    // receive message
[mess] r=1 & r_even=1 & mess<N-1 -> (r'=1) & (mess'=min(mess+1,N)) &
(r_even'=0);
    // receive message and last
[mess] r=1 & r_even=1 & mess=N-1 -> (r'=3) & (mess'=min(mess+1,N));
[done] r=3 -> true; // finished so loop
Endmodule

```

Figure 8.2: A fragment of the PRISM code for Version 5

From figure 8.2 (with comments inclusive) above, assuming the system can either be in an even or odd phase, **o_even : [0..1]** and **r_even : [0..1]** are variables used to identify an even or odd state of the model. **0** represents an odd state and **1** represents an even state. Both parties behave differently when in either an even or in an odd state. This state's change frequently depending on if the guards in the commands are satisfied. At the start of the protocol run **o_even** and **r_even** are initially 0 and the originator starts by sending a message to the recipient. He receives an evidence afterwards and the states change to 1 (an even state). In an even state, the recipient sends evidence first to the originator before he can receive a message. Since the steps of the protocol run are either in an even or odd phase, then this can be used for subsequent steps to decide who sends an item first.

The simulation of this version in PRISM is seen in figure 8.2a and b below, it is observed that when both parties exchange items first interchangeably, then the unfairness rate for both parties becomes balanced (i.e. the protocol is no longer unfair to only one party for a long period of time). In figure 8.2a and 8.2b below, even if the originator has just sent a message, the protocol is still very much fair to both parties.

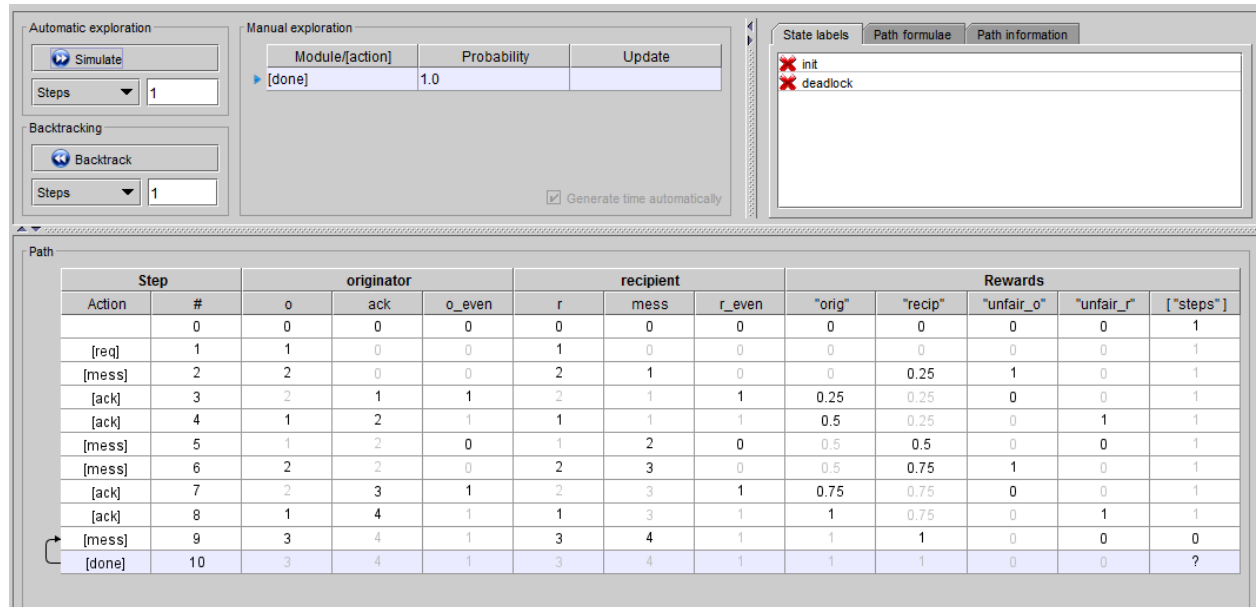


Figure 8.2a: Simulation of deterministic model version 5 in PRISM when $n=4$ (i.e. even)

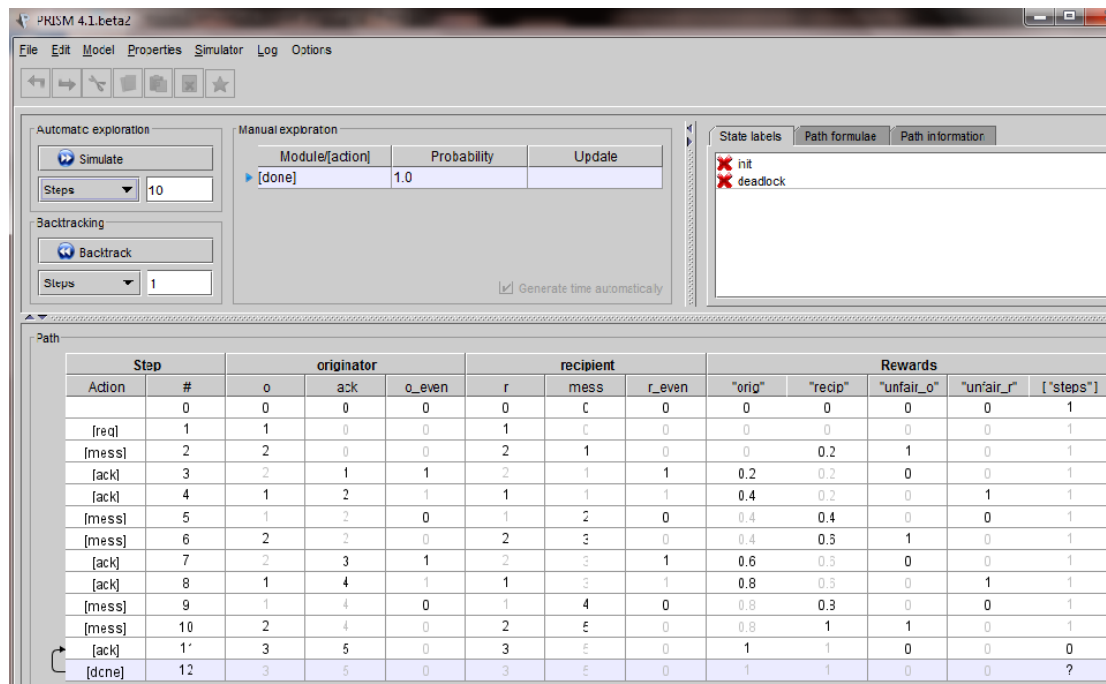


Figure 8.2b: Simulation of deterministic model version 5 in PRISM when $n=5$ (i.e. odd)

8.2 Analysis of Version 5 in PRISM

8.2.1 Probability that version 5 terminates correctly

This version terminates correctly with probability equal to 1 like the other versions of the models.

8.2.2 Expected evidence each party has after k steps in version 5

Both parties have equal opportunities to receive their evidence first (if n is even). This makes the graph analyzing their expected evidence as follows:

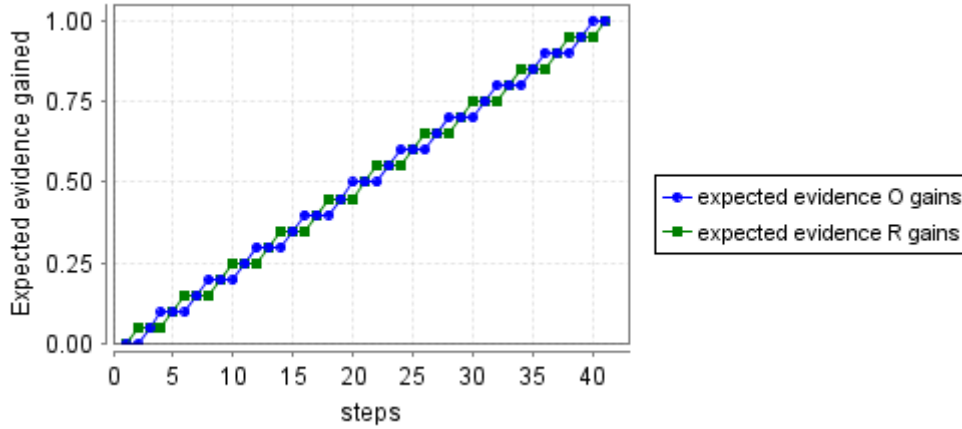


Figure 8.3a: Expected evidence each party gains when $n=20$ (deterministic model)

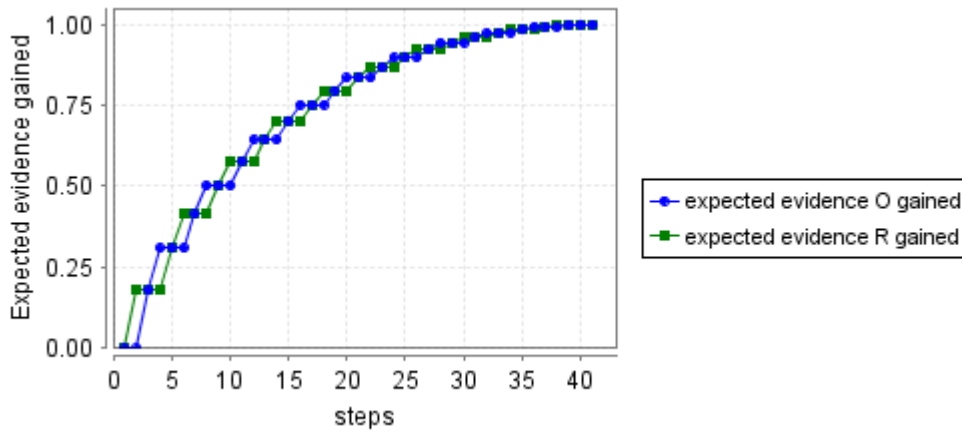


Figure 8.3b: Expected evidence each party gains when n is chosen randomly from 1...20 in the probabilistic model

With probability equal to 1 both parties receive their expected evidence at the end of the protocol run thus making this model viable.

8.2.3 Probability of unfairness in version 5

The probability of the protocol being unfair to either party can be seen in figure 8.4a and 8.4b below. The graph pattern appears the way it is because both

parties get to send evidence first at some point during the protocol run which increases the chances of the protocol being equally fair.

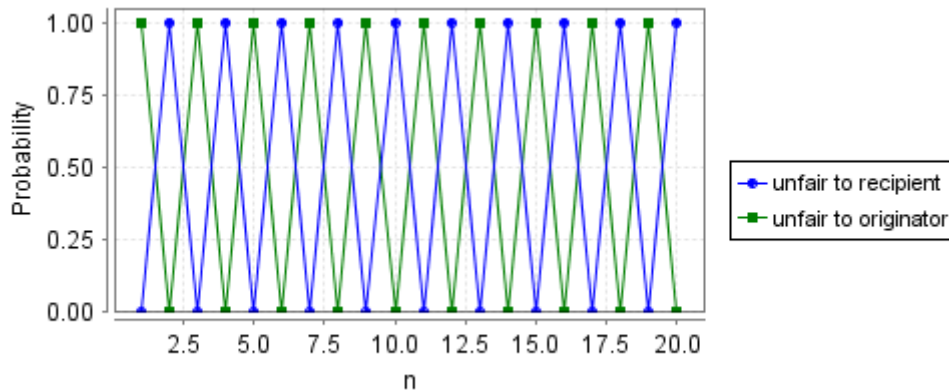


Figure 8.4a: Probability of unfairness when n varies from 1...20 (deterministic model)

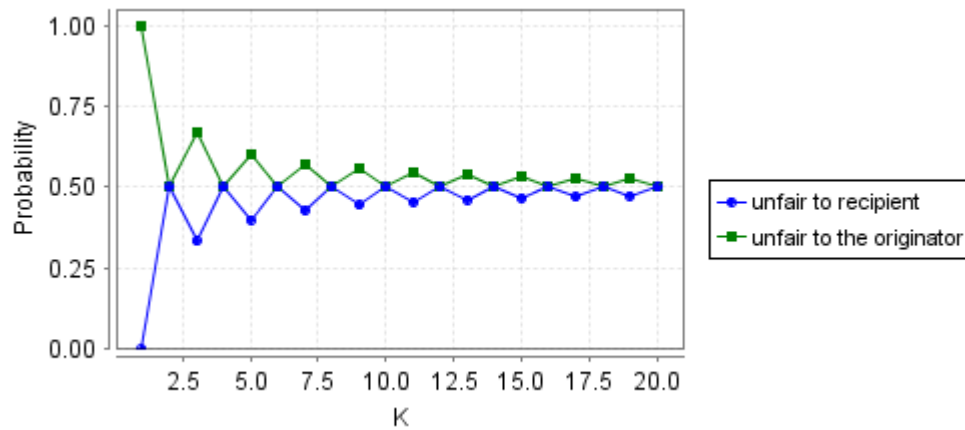


Figure 8.4b: Probability of unfairness when n chosen randomly from 1...20 (probabilistic model)

From figure 8.4a above, it is observed that in version 5, if n is an even number, then the protocol is equally fair (and unfair) to both parties. On the other hand, if n is an odd number, the originator is at a disadvantage with regards to fairness. However, from figure 8.4b above, we can observe that as K increases, the chances of being equally fair for odd values of K increases (i.e. the difference decreases). The fact that the probability of unfairness is reduced in this new version makes this new ordering of messages a better option and an improvement for fairness.

8.2.4 Time each party spends in an unfair state in version 5

During the protocol run, the time each party spends in an unfair state is seen in figure 8.5 below, it is observed that with this new way of sending messages, both parties tend to spend about the same (lesser) amount of time in an unfair state than in the previous versions of the models.

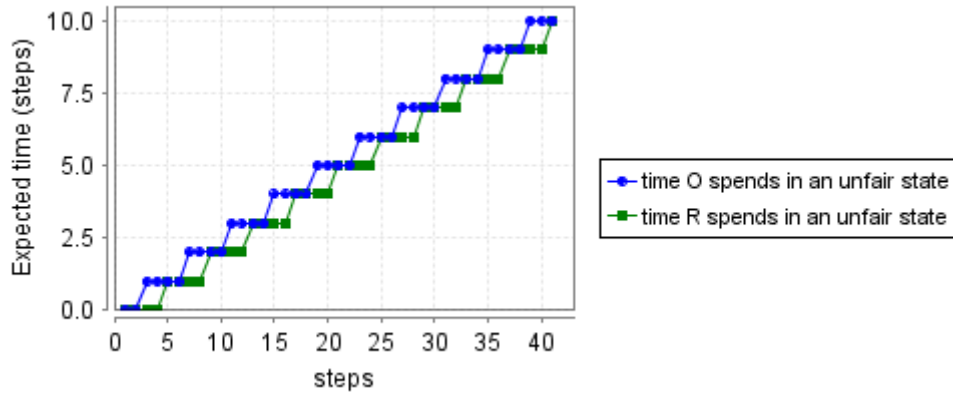


Figure 8.5a: Time each party spends in an unfair state when $n=20$ (deterministic model)

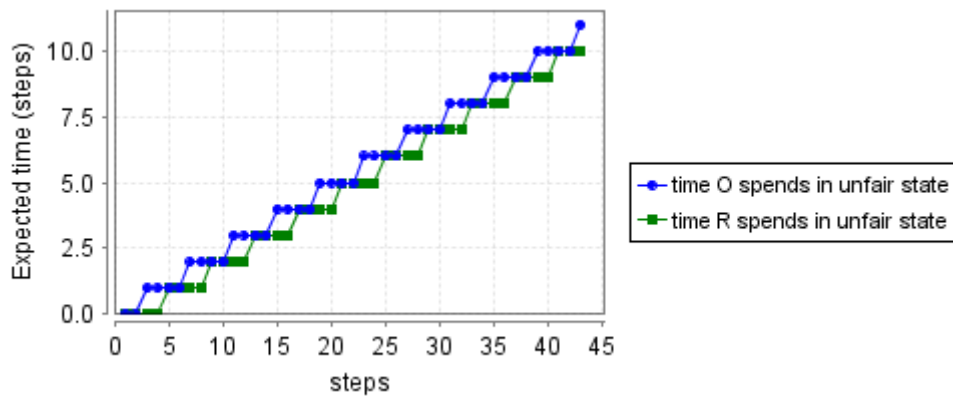


Figure 8.5b: Time each party spends in an unfair state when $n=21$ (deterministic model)

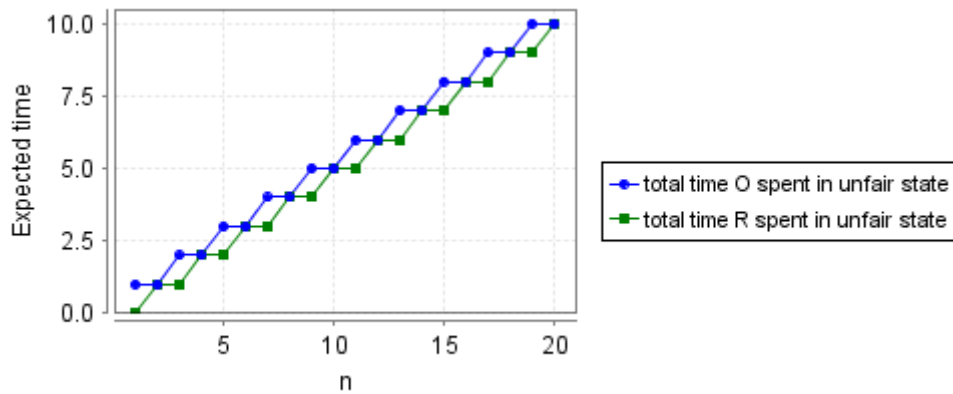


Figure 8.5c: Total time both parties spent in an unfair state when $n=20$

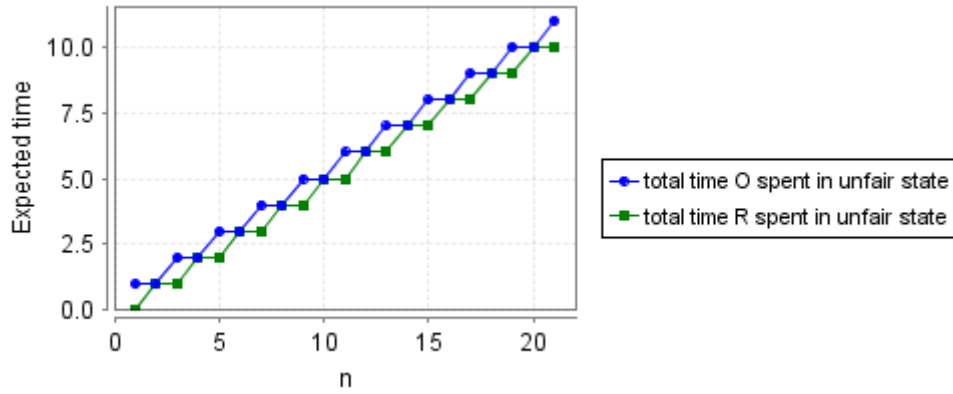


Figure 8.5d: Total time both parties spent in an unfair state when $n=21$

From figure 8.5 a,b,c & d above, it can be observed that if n is even then both parties spend the same time in an unfair state but if n is odd then the originator spends more time in an unfair state.

Figure 8.4e and 8.4f shows the time spent by both parties in an unfair state in the probabilistic model. From the figures, we can see that O spends more time in an unfair state and is at a slight disadvantage because he has to go first (see figure 8.4f below).

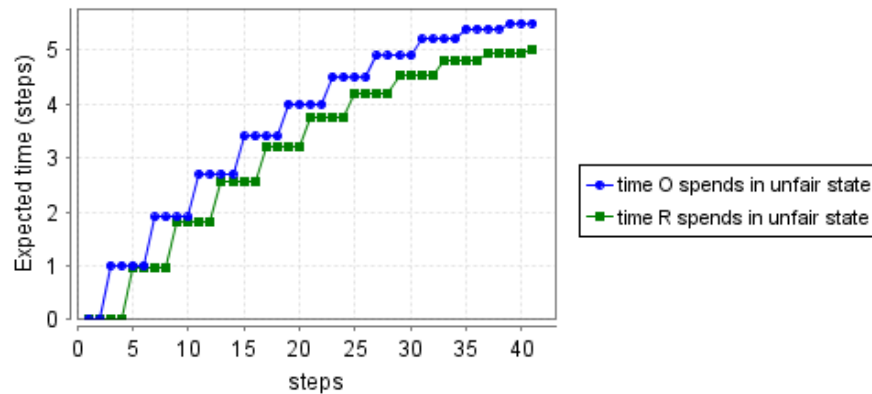


Figure 8.4e: Time both parties spend in an unfair state when $K=20$ and n is chosen randomly from $1...K$ (probabilistic model)

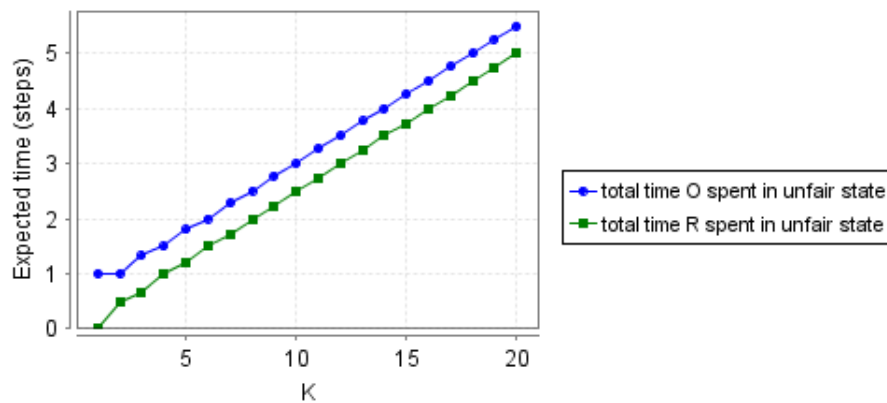


Figure 8.4f: Total time both parties spent in an unfair state when $K=20$ and n chosen randomly from $1...K$ (probabilistic model)

8.2.5 Expected time (number of steps) to complete the protocol run in version 5

The number of steps required for this version of the model is the same as previous versions of the model.

8.3 Conclusion

Who goes first in providing the evidence required by the other party is achieved interchangeably at each step.

This version terminates correctly with a probability equal to 1, both parties receive their expected evidences incrementally and with probability equal to 1 it is viable (1). When \mathbf{n} or \mathbf{K} is even, both parties spent the same time in an unfair state while when \mathbf{n} is odd the originator spent more time in an unfair state. It is observed that O is at a slight disadvantage because he has to go first at the start of the protocol run.

It is observed that this version gives a 50 % chance improvement of fairness being attained as opposed to 0% chance of fairness for the originator at each step (subsection 4.3.3) from the original specification of the protocol in (1). However, when \mathbf{K} (the interval from which \mathbf{n} is chosen randomly) in the probabilistic model is odd, the probability of attaining equal fairness for both parties can be achieved as \mathbf{K} increases. This makes version 5 a great improvement to the unfairness issue discovered.

Chapter 9 Conclusion and Future Work

Whether both parties to the protocol intends to act honestly or not, the non-repudiation protocol without a trusted third party as proposed by Markowitch & Roggeman in (1) has been found to have some problems with regards to the fairness property (it is unfair to the originator with probability equal to 1 and unfair to the recipient with probability equal to 0). These problems are, in particular, due to the fact that

- the recipient always gets messages first and
- the originator gets nothing towards his NRR evidence until he receives the final acknowledgement while the recipient gets things incrementally that makes up his NRO evidence.

To resolve these problems found and improve fairness, both parties are modeled to receive their evidence incrementally at each step. This evidence can be presented to a judge in case of a dispute when the recipient tries to cheat the originator. Since the acknowledgement is signed by the recipient and time stamped, then in a case when the recipient tries to cheat/repudiate, the originator can present this evidence and the last message he sent after that, after this the judge checks the evidence and can conclude that the recipient indeed took part in the communication. This structure of accumulating evidences for both parties reduces the possibility of a denial on the part of both parties.

The message ordering was also changed in such a way that both parties exchange items first in an interchangeable manner. A change in the ordering of the messages so that the originator can receive an evidence first before sending a message at some point during the protocol run will help to reduce the disadvantage of always sending a message first the originator has. I argue that this change is feasible because it reflects common e-commerce practices where a “payment before service” notion is popular. Moreover a similar approach was taken in order to improve the fairness of the probabilistic contract signing protocol of Even, Goldreich and Lempel (EGL protocol) in (25). In their approach (which is similar to my proposed approach for improving fairness), the authors investigated (using PRISM) how the EGL protocol can be made fairer by changing the message ordering of the EGL protocol (this actually helped in improving the fairness of the EGL protocol).

These two changes were made to the basic protocol proposed by Markowitch and Roggeman and they have been modeled and analyzed in different versions using the PRISM tool.

Two models of the protocol are considered. The first is a **deterministic model** and the second a **probabilistic model**. These models are considered for simplicity and explanation of the unfairness problems of the protocol stated. The deterministic model, considers the simplest scenario of when both parties make no other choice than to behave honestly, follow the protocol accordingly until it terminates successfully. In the deterministic model, n is fixed in advance (n represents the number of *evidence* that needs to be transmitted). While in the probabilistic model of the protocol developed a parameter K is introduced and n is uniformly distributed over the interval $1, \dots, K$. This is similar to the

specification of the protocol which states that the originator chooses n secrets from a geometric distribution. Different versions of the models were considered and through thorough and effective model checking algorithms developed in PRISM modeling language, a number of properties and rewards were used to analyze the fairness, timeliness and viability properties of the different versions.

From the analysis of the different versions, it has been observed that with the new changes implemented, there is a 50% chance of fairness to both parties. This is the best that can be hoped for because someone has to always start first by sending a message to the other party. However, it is important to note that there is a 50/50 chance of fairness for the originator and recipient if K is even. Also, at the first step of the protocol run i.e. when $n = 1$, the protocol is always unfair to the originator because he has to always respond to a request by sending a message first. Therefore, it is advisable that n should never be 1 and that K be an even number from which n is chosen randomly.

The value of n chosen by the originator is directly proportional to the time spent for the protocol run. And since a lesser time/number of steps required completing the protocol run means better performance of the protocol (27). Then it is advisable that n be kept as small as possible if good performance is desired.

9.1 Future Work

As a result of a time constraint limitation on this project, this project was restricted to discrete steps and the models did not consider real time or non-deterministic behaviors. However, some future works are suggested in this section.

9.1.1 PTA Modeling and Analysis

The initial intention of this project was to design and analyze a probabilistic timed automata (PTA) model that incorporates non-deterministic, probabilistic and real time behaviors of the non-repudiation protocol proposed by Markowitch and Roggeman in (1). However, the unfairness problem discovered with the protocol had to be resolved first. So therefore, in the future the new changes made to improve fairness can be implemented while designing PTA models that combines and analyzes the non-deterministic, probabilistic and real time behaviors of the protocol.

Probabilistic timed automata are formal models for systems that have both probabilistic, non-deterministic and real time behaviors (4). PTAs are a finite state automata extended with real-valued clocks and discrete probabilistic choice. It has real-valued model of time. PTAs analyses a wide range of properties ranging from reliability to performance of probability functions (e.g. “minimum probability that a packet will deliver at a set time” and “maximum probability that a protocol will fail to execute within specified time”) or time functions (e.g. “minimum time expected for the protocol to execute completely” or “maximum time expected for the protocol to terminate”). The resulting model when additional information in the form of cost or rewards is added to a PTA is known as Priced Probabilistic Timed Automata (PPTAs). These PPTAs analyze a wider range of properties (e.g. “maximum expected number of lost packets within the first minute”). A fundamental property of PTA is the minimum and maximum

probability of reaching a particular set of states in the model. PTAs are useful for modeling the protocol because there is probabilistic behavior (choice of n), real time behavior (time to send the messages and acknowledgements, deadline and time to compose the messages) and non-deterministic behaviors (choice of when or if to send messages or acknowledgements).

Therefore, there should be a development and formal analysis of a concrete model that combines probabilistic, non-deterministic & real time behaviors. The simple PTA models developed in (4) can be considered as a starting point. In (4) two properties of the protocol (i.e. bounded by time and fairness) were considered (the viability property was not considered). However, both the models and analysis was limited, as they were used only to show the feasibility of PTA model checking.

9.1.2 Applying stochastic two player games

The interactions between the two parties in the protocol can be modeled by applying stochastic two player games (18) (28). Games have a structure where two or more players are trying to compete against each other for something e.g. a reward or a goal. A player has a number of choices that they can use to obtain their reward or goal. And one player e.g. Alice might decide to make a move in order to have some advantage over another player e.g. Bob (i.e. Alice may try to cheat Bob). We can consider the two parties involved in a non-repudiation protocol (i.e. the originator and recipient) as players in a game setting, since the two parties of the protocol have separate goals (the originator's goal is to receive the NRR and the recipient's goal is to receive the message and the NRO). The parties compete with each other to receive their rewards (i.e. their expected evidences), and both have different choices that they can make at each step of the protocol's execution to obtain their rewards and one party e.g. the recipient might want to take advantage of the originator just as Alice may try to cheat Bob. This game structure makes it an interesting technique to use in modeling and analyzing the interactive behaviors between the two parties of the protocol. PRISM has an extension to support games called PRISM-games (7) (28). This can be used as the tool for checking the game model.

9.1.3 Detailed Security and Performance analysis

A more detailed security and performance analysis other than the once done in this project or in (27) should be carried out. To achieve this, not only will the parties of the protocol be modeled, but also the communication channel between the two parties will have to be modeled as well.

9.1.4 Extending the protocol to a Multiparty Scenario

This project considers a simple security scenario in which two parties need to exchange information items. But in many cases, in real applications, we have a scenario where multiple entities are participating (this is called a multiparty scenario). Therefore, the protocol can be modeled and analyzed to accommodate such scenarios. This would give a Multiparty Non-repudiation protocol (MNR).

Appendix A Prism Codes and simulation of models

A. Property Specification Codes

```
const int k; // used in experiments for number of steps

// probability protocol terminates correctly
P=?[F o=3 & r=3]

R{"orig"}=? [I=k] // contribution originator has to NRR after k
steps
R{"recip"}=? [I=k] // contribution repudiation has to NRO after k
steps

P=?[ F (ack=N) & (mess<N) ] // unfair to recipient
P=?[ F (ack<N) & (mess=N) ] // unfair to originator

// time in an unfair state (after k steps and when finished)
R{"unfair_o"}=? [C<=k]
R{"unfair_r"}=? [C<=k]
R{"unfair_o"}=? [F o=3 & r=3]
R{"unfair_r"}=? [F o=3 & r=3]

// expected time to complete the protocol
R{"steps"}=?[F o=3 & r=3]
```

Figure A: PRISM Property specification code for the models

A.1 Version 1: PRISM codes and simulations

```
dtmc // model is a dtmc

const int K; // maximum number of messages
// number of messages uniformly chosen over 1,...,K

// labels for unfairness
formula unfair_o = (ack<N & mess=N); // unfair to originator
formula unfair_r = (ack=N & mess<N); // unfair to recipient

module originator

    o : [0..3]; // local state of originator
    // 0 initial state
    // 1 send messages
    // 2 receive acks
    // 3 done
    N : [0..K]; // number of messages
    ack : [0..K]; // number of acks the originator has received

    // get request (so set K)
    [req] o=0 & K=1 -> 1/K : (o'=1) & (N'=1);
    [req] o=0 & K=2 -> 1/K : (o'=1) & (N'=1)
        + 1/K : (o'=1) & (N'=2);
```

```

[req] o=0 & K=3 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3);

[req] o=0 & K=4 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4);

[req] o=0 & K=5 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5);

[req] o=0 & K=6 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6);

[req] o=0 & K=7 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7);

[req] o=0 & K=8 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7)
      + 1/K : (o'=1) & (N'=8);

[req] o=0 & K=9 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7)
      + 1/K : (o'=1) & (N'=8)
      + 1/K : (o'=1) & (N'=9);

[req] o=0 & K=10 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7)
      + 1/K : (o'=1) & (N'=8)
      + 1/K : (o'=1) & (N'=9)
      + 1/K : (o'=1) & (N'=10);

[mess] o=1 -> (o'=2); // send message
[ack] o=2 & ack<N-1 -> (o'=1) & (ack'=min(ack+1,K)); // receive
ack & not last
[ack] o=2 & ack=N-1 -> (o'=3) & (ack'=min(ack+1,K)); // receive
last ack

```

```

    [done] o=3 -> true; // finished so loop

endmodule

module recipient

    r : [0..3]; // local state of recipient
    // 0 initial state
    // 1 receive messages
    // 2 send acks
    // 3 done
    mess : [0..K]; // number of mess the originator has received

    [req] r=0 -> (r'=1); // send request
    [mess] r=1 -> (r'=2) & (mess' = min(mess+1, K)); // receive message
    [ack] r=2 & mess<N -> (r'=1); // send ack & not last
    [ack] r=2 & mess=N -> (r'=3); // send last ack
    [done] r=3 -> true; // finished so loop

endmodule

rewards "orig" // contribution to KRR the originator has
    ack=N : 1;
endrewards

rewards "recip" // contribution to KRO the recipient has
    true : mess/N;
endrewards

rewards "unfair_o" //in an unfair state (for originator)
    mess>0 & ack<N : 1;
endrewards

rewards "unfair_r" //in an unfair state (for recipient)
    ack=N & mess<N : 1;
endrewards

rewards "steps" // number steps (time)
    [req] true : 1;
    [mess] true : 1;
    [ack] true : 1;
endrewards

```

Figure A.1.1a: Probabilistic model PRISM codes

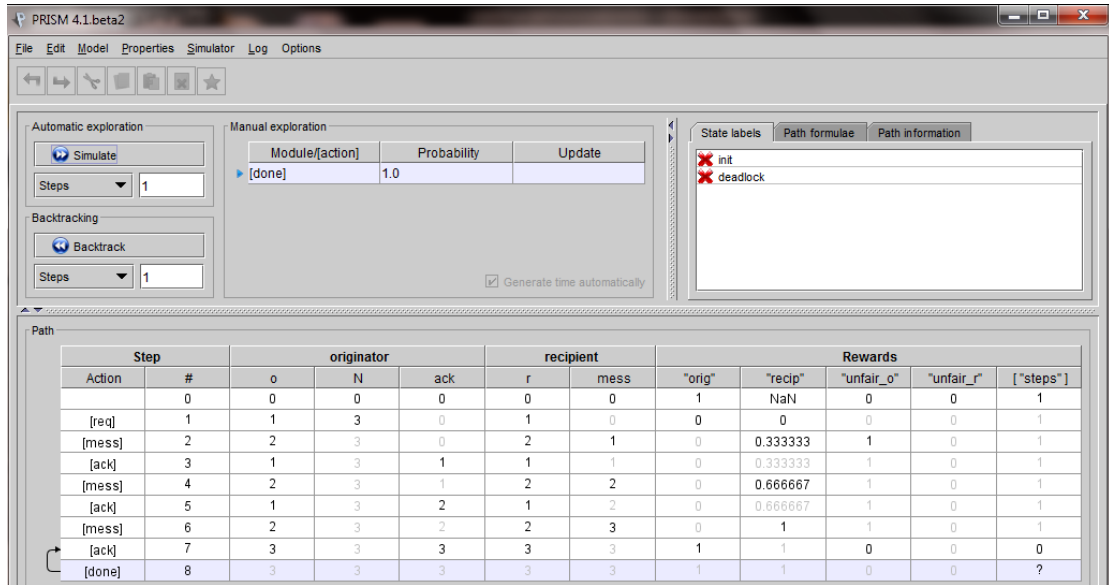


Figure A.1.1b: Probabilistic model simulation when n is randomly chosen as 3

A.2 Version 2: PRISM code and simulation

```
dtmc // model is a dtmc

const int N; // fix N in advance

// labels for unfairness
formula unfair_o = (ack<N & mess=N); // unfair to originator
formula unfair_r = (ack=N & mess<N); // unfair to recipient

module originator

    o : [0..3]; // local state of originator
    // 0 initial state
    // 1 send messages
    // 2 receive acks
    // 3 done
    ack : [0..N]; // number of acks the originator has received

    [req] o=0 -> (o'=1); // get request
    [mess] o=1 -> (o'=2); // send message
    [ack] o=2 & ack<N-1 -> (o'=1) & (ack'=min(ack+1,N)); // receive
ack & not last
    [ack] o=2 & ack=N-1 -> (o'=3) & (ack'=min(ack+1,N)); // receive
last ack
    [done] o=3 -> true; // finished so loop
endmodule

module recipient

    r : [0..3]; // local state of originator
    // 0 initial state
    // 1 receive messages
```

```

// 2 send acks
// 3 done
mess : [0..N]; // number of mess the originator has received

[req] r=0 -> (r'=1); // send request
[mess] r=1 -> (r'=2) & (mess'==min(mess+1,N)); // receive message
[ack] r=2 & mess<N -> (r'=1); // send ack & not last
[ack] r=2 & mess=N -> (r'=3); // send last ack
[done] r=3 -> true; // finished so loop

endmodule

rewards "orig" // contribution to NRR the originator has
  true : ack/N;
endrewards

rewards "recip" // contribution to NRO the recipient has
  true : mess/N;
endrewards

rewards "unfair_o" //in an unfair state (for originator)
  ack<mess : 1;
endrewards

rewards "unfair_r" //in an unfair state (for recipient)
  mess<ack : 1;
endrewards

rewards "steps" // number steps (time)
  [req] true : 1;
  [mess] true : 1;
  [ack] true : 1;
endrewards

```

Figure A.2.1a: Deterministic model PRISM code

The screenshot shows the PRISM 4.1 beta2 interface. The 'Manual exploration' tab is selected, displaying a table of simulation steps. The table has columns for Step, Action, #, originator (o, ack), recipient (r, mess), and various rewards (orig, recip, unfair_o, unfair_r, steps). The simulation starts at 'init' and proceeds through a series of 'req', 'mess', and 'ack' actions, eventually reaching a 'done' state at step 14.

Step	Action	#	originator		recipient		Rewards				
			o	ack	r	mess	"orig"	"recip"	"unfair_o"	"unfair_r"	["steps"]
0		0	0	0	0	0	0	0	0	1	
1	[req]	1	1	0	1	0	0	0	0	1	
2	[mess]	2	2	0	2	1	0	0.166667	1	0	
3	[ack]	3	1	1	1	1	0.166667	0.166667	0	1	
4	[mess]	4	2	1	2	2	0.166667	0.333333	1	0	
5	[ack]	5	1	2	1	2	0.333333	0.333333	0	1	
6	[mess]	6	2	2	2	3	0.333333	0.5	1	0	
7	[ack]	7	1	3	1	3	0.5	0.5	0	1	
8	[mess]	8	2	3	2	4	0.5	0.666667	1	0	
9	[ack]	9	1	4	1	4	0.666667	0.666667	0	1	
10	[mess]	10	2	4	2	5	0.666667	0.833333	1	0	
11	[ack]	11	1	5	1	5	0.833333	0.833333	0	1	
12	[mess]	12	2	5	2	6	0.833333	1	1	0	
13	[ack]	13	3	6	3	6	1	1	0	0	
14	[done]	14	3	6	3	6	1	1	0	?	

Figure A.2.1b: Deterministic model simulation when n=6

```
dtmc // model is a dtmc

const int K; // maximum number of messages
// number of messages uniformly chosen over 1,...,K

// labels for unfairness
formula unfair_o = (ack<N & mess=N); // unfair to originator
formula unfair_r = (ack=N & mess<N); // unfair to recipient

module originator

  o : [0..3]; // local state of originator
  // 0 initial state
  // 1 send messages
  // 2 receive acks
  // 3 done
  N : [0..K]; // number of messages
  ack : [0..K]; // number of acks the originator has received

  // get request (so set K)
  [req] o=0 & K=1 -> 1/K : (o'=1) & (N'=1);
  [req] o=0 & K=2 -> 1/K : (o'=1) & (N'=1)
    + 1/K : (o'=1) & (N'=2);
  [req] o=0 & K=3 -> 1/K : (o'=1) & (N'=1)
    + 1/K : (o'=1) & (N'=2)
    + 1/K : (o'=1) & (N'=3);

  [req] o=0 & K=4 -> 1/K : (o'=1) & (N'=1)
    + 1/K : (o'=1) & (N'=2)
    + 1/K : (o'=1) & (N'=3)
    + 1/K : (o'=1) & (N'=4);
  [req] o=0 & K=5 -> 1/K : (o'=1) & (N'=1)
    + 1/K : (o'=1) & (N'=2)
    + 1/K : (o'=1) & (N'=3)
    + 1/K : (o'=1) & (N'=4)
    + 1/K : (o'=1) & (N'=5);
  [req] o=0 & K=6 -> 1/K : (o'=1) & (N'=1)
    + 1/K : (o'=1) & (N'=2)
    + 1/K : (o'=1) & (N'=3)
    + 1/K : (o'=1) & (N'=4)
    + 1/K : (o'=1) & (N'=5)
    + 1/K : (o'=1) & (N'=6);
  [req] o=0 & K=7 -> 1/K : (o'=1) & (N'=1)
    + 1/K : (o'=1) & (N'=2)
    + 1/K : (o'=1) & (N'=3)
    + 1/K : (o'=1) & (N'=4)
    + 1/K : (o'=1) & (N'=5)
    + 1/K : (o'=1) & (N'=6)
    + 1/K : (o'=1) & (N'=7);
  [req] o=0 & K=8 -> 1/K : (o'=1) & (N'=1)
    + 1/K : (o'=1) & (N'=2)
    + 1/K : (o'=1) & (N'=3)
    + 1/K : (o'=1) & (N'=4)
    + 1/K : (o'=1) & (N'=5)
    + 1/K : (o'=1) & (N'=6)
    + 1/K : (o'=1) & (N'=7)
```

```

      + 1/K : (o'=1) & (N'=8);
[req] o=0 & K=9 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7)
      + 1/K : (o'=1) & (N'=8)
      + 1/K : (o'=1) & (N'=9);
[req] o=0 & K=10 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7)
      + 1/K : (o'=1) & (N'=8)
      + 1/K : (o'=1) & (N'=9)
      + 1/K : (o'=1) & (N'=10);
[req] o=0 & K=11 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7)
      + 1/K : (o'=1) & (N'=8)
      + 1/K : (o'=1) & (N'=9)
      + 1/K : (o'=1) & (N'=10)
      + 1/K : (o'=1) & (N'=11);
[req] o=0 & K=12 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7)
      + 1/K : (o'=1) & (N'=8)
      + 1/K : (o'=1) & (N'=9)
      + 1/K : (o'=1) & (N'=10)
      + 1/K : (o'=1) & (N'=11)
      + 1/K : (o'=1) & (N'=12);
[req] o=0 & K=13 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7)
      + 1/K : (o'=1) & (N'=8)
      + 1/K : (o'=1) & (N'=9)
      + 1/K : (o'=1) & (N'=10)
      + 1/K : (o'=1) & (N'=11)
      + 1/K : (o'=1) & (N'=12)
      + 1/K : (o'=1) & (N'=13);
[req] o=0 & K=14 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)

```



```

+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14);
[req] o=0 & K=15 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15);
[req] o=0 & K=16 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15)
+ 1/K : (o'=1) & (N'=16);
[req] o=0 & K=17 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15)

```

```

+ 1/K : (o'=1) & (N'=16)
+ 1/K : (o'=1) & (N'=17);
[req] o=0 & K=18 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15)
+ 1/K : (o'=1) & (N'=16)
+ 1/K : (o'=1) & (N'=17)
+ 1/K : (o'=1) & (N'=18);
[req] o=0 & K=19 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15)
+ 1/K : (o'=1) & (N'=16)
+ 1/K : (o'=1) & (N'=17)
+ 1/K : (o'=1) & (N'=18)
+ 1/K : (o'=1) & (N'=19);
[req] o=0 & K=20 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15)
+ 1/K : (o'=1) & (N'=16)
+ 1/K : (o'=1) & (N'=17)
+ 1/K : (o'=1) & (N'=18)
+ 1/K : (o'=1) & (N'=19)

```

```

        + 1/K : (o'=1) & (N'=20);
    [mess] o=1 -> (o'=2); // send message
    [ack] o=2 & ack<N-1 -> (o'=1) & (ack'=min(ack+1,K)); // receive
ack & not last
    [ack] o=2 & ack=N-1 -> (o'=3) & (ack'=min(ack+1,K)); // receive
last ack
    [done] o=3 -> true; // finished so loop

endmodule

module recipient

    r : [0..3]; // local state of recipient
    // 0 initial state
    // 1 receive messages
    // 2 send acks
    // 3 done
    mess : [0..K]; // number of mess the originator has received

    [req] r=0 -> (r'=1); // send request
    [mess] r=1 -> (r'=2) & (mess'=min(mess+1,K)); // receive message
    [ack] r=2 & mess<N -> (r'=1); // send ack & not last
    [ack] r=2 & mess=N -> (r'=3); // send last ack
    [done] r=3 -> true; // finished so loop

endmodule

rewards "orig" // contribution to KRR the originator has
    true : ack/N;
endrewards

rewards "recip" // contribution to KRO the recipient has
    true : mess/N;
endrewards

rewards "unfair_o" //in an unfair state (for originator)
    ack<mess : 1;
endrewards

rewards "unfair_r" //in an unfair state (for recipient)
    mess<ack : 1;
endrewards

rewards "steps" // number steps (time)
    [req] true : 1;
    [mess] true : 1;
    [ack] true : 1;
endrewards

```

Figure A.2.2a: Probabilistic model PRISM code (V2)

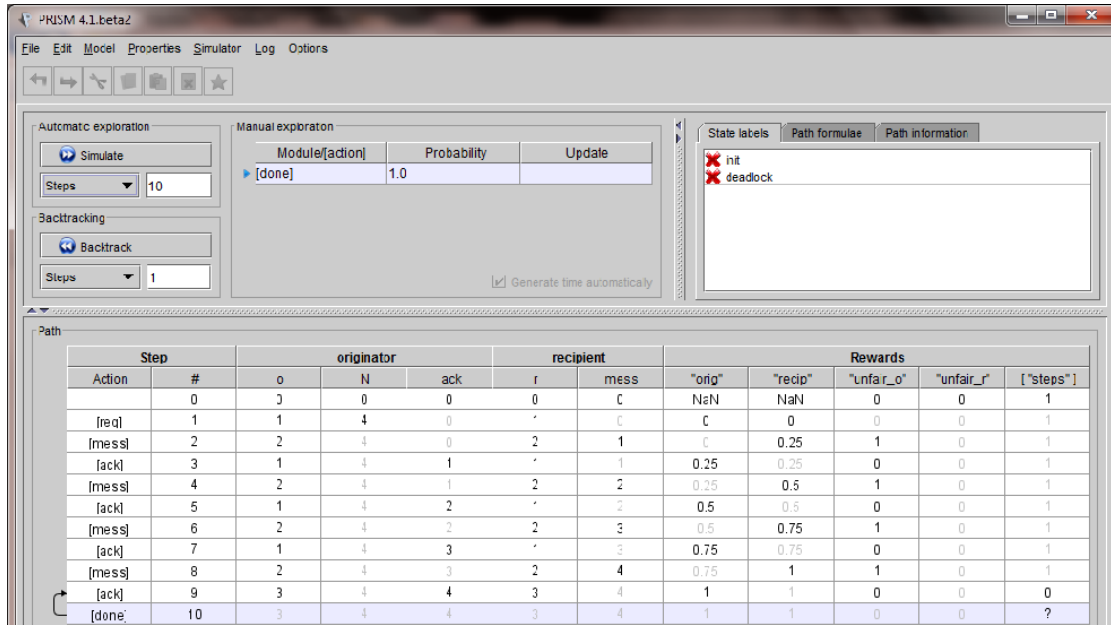


Figure A.2.2b: Probabilistic model simulation when n is randomly chosen as 4 (V2)

A.3 Version 3: PRISM codes and simulation

```
dtmc // model is a dtmc

const int N; // fix N in advance

// labels for unfairness
formula unfair_o = (ack<N & mess=N); // unfair to originator
formula unfair_r = (ack=N & mess<N); // unfair to recipient

module originator
  o : [0..3]; // local state of originator

  // 0 initial state

  // 1 send messages

  // 2 receive acks

  // 3 done

  ack : [0..N]; // number of acks the originator has received

  [req] o=0 -> (o'=1); // get request
  [mess] o=1 & ack<N -> (o'=2); // send message
  [mess] o=1 & ack=N -> (o'=3); // send last message
  [ack] o=2 & ack<N-2 -> (o'=1) & (ack'=min(ack+1,N)); // receive
  ack & not last but one ack
```

```

    [ack] o=2 & ack=N-2 -> (o'=2) & (ack'=min(ack+1,N)); // receive
ack & last but one (so now wait for ack before sending last
message)
    [ack] o=2 & ack=N-1 & N>1 -> (o'=1) & (ack'=min(ack+1,N)); //
receive last ack and N>1
    [ack] o=2 & ack=N-1 & N=1 -> (o'=3) & (ack'=min(ack+1,N)); //
receive last ack and N=1
    [done] o=3 -> true; // finished so loop
endmodule

```

```

module recipient
  r : [0..3]; // local state of originator

  // 0 initial state
  // 1 receive messages
  // 2 send acks
  // 3 done
  mess : [0..N]; // number of mess the originator has received
  counter : [0..1];

  [req] r=0 -> (r'=1); // sends request
  [mess] r=1 & mess<N-1 -> (r'=2) & (mess'=min(mess+1,N)); //
receive message
  [mess] r=1 & mess=0 & N=1 -> (r'=2) & (mess'=min(mess+1,N)); //
receive last message and N=1
  [mess] r=1 & mess=N-1 & N>1 -> (r'=3) & (mess'=min(mess+1,N));
// receive last message and N>1
  [ack] r=2 & mess<N-1 -> (r'=1); // send ack & not last but one
  [ack] r=2 & N=1 -> (r'=3); // send last ack and N=1
  [ack] r=2 & N>1 & mess=N-1 & counter=0 -> (r'=2) & (counter'=1);
// send last but one ack and N>1
  [ack] r=2 & N>1 & mess=N-1 & counter=1 -> (r'=1) & (counter'=0);
// send last ack and N>1
  [done] r=3 -> true; // finished so loop
Endmodule

```

```

rewards "orig" // contribution to NRR the originator has
  true : ack/N;
endrewards

```

```

rewards "recip" // contribution to NRO the recipient has
  true : mess/N;
endrewards

```

```

rewards "unfair_o" //in an unfair state (for originator)
  ack<mess : 1;
endrewards

```

```

rewards "unfair_r" //in an unfair state (for recipient)
  mess<ack : 1;
endrewards

```

```

rewards "steps" // number steps (time)
  [req] true : 1;

```

```

[mess] true : 1;
[ack] true : 1;
endrewards

```

Figure A.3.1a: Deterministic model Prism code (V3)

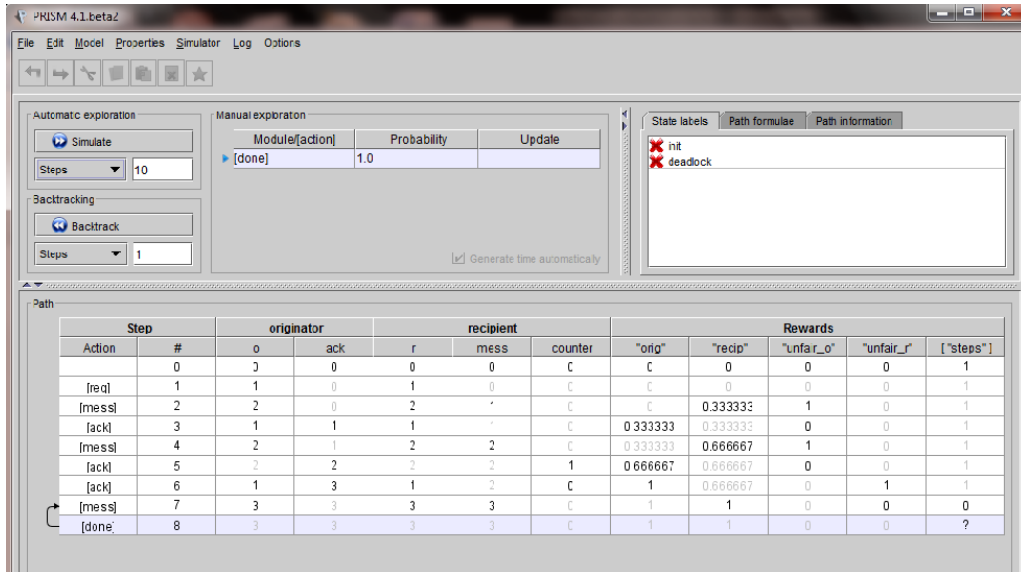


Figure A.3.1b: Deterministic model simulation when n is 3 (V3)

```

dtmc // model is a dtmc

const int K; // maximum number of messages
// number of messages uniformly chosen over 1,...,K

// labels for unfairness
formula unfair_o = (ack<N & mess=N); // unfair to originator
formula unfair_r = (ack=N & mess<N); // unfair to recipient

module originator

    o : [0..3]; // local state of originator
    // 0 initial state
    // 1 send messages
    // 2 receive acks
    // 3 done
    N : [0..K]; // number of messages
    ack : [0..K]; // number of acks the originator has received

    // get request (so set K)
    [req] o=0 & K=1 -> 1/K : (o'=1) & (N'=1);
    [req] o=0 & K=2 -> 1/K : (o'=1) & (N'=1)
        + 1/K : (o'=1) & (N'=2);
    [req] o=0 & K=3 -> 1/K : (o'=1) & (N'=1)
        + 1/K : (o'=1) & (N'=2)
        + 1/K : (o'=1) & (N'=3);

    [req] o=0 & K=4 -> 1/K : (o'=1) & (N'=1)
        + 1/K : (o'=1) & (N'=2)
        + 1/K : (o'=1) & (N'=3)
        + 1/K : (o'=1) & (N'=4);
    [req] o=0 & K=5 -> 1/K : (o'=1) & (N'=1)

```

```

+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5);
[req] o=0 & K=6 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6);
[req] o=0 & K=7 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7);
[req] o=0 & K=8 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8);
[req] o=0 & K=9 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9);
[req] o=0 & K=10 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10);
[req] o=0 & K=11 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11);
[req] o=0 & K=12 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)

```

```

+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12);
[req] o=0 & K=13 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13);
[req] o=0 & K=14 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14);
[req] o=0 & K=15 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15);
[req] o=0 & K=16 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)

```



```

+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15)
+ 1/K : (o'=1) & (N'=16);
[req] o=0 & K=17 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15)
+ 1/K : (o'=1) & (N'=16)
+ 1/K : (o'=1) & (N'=17);
[req] o=0 & K=18 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15)
+ 1/K : (o'=1) & (N'=16)
+ 1/K : (o'=1) & (N'=17)
+ 1/K : (o'=1) & (N'=18);
[req] o=0 & K=19 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)

```

```

+ 1/K : (o'=1) & (N'=15)
+ 1/K : (o'=1) & (N'=16)
+ 1/K : (o'=1) & (N'=17)
+ 1/K : (o'=1) & (N'=18)
+ 1/K : (o'=1) & (N'=19);
[req] o=0 & K=20 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15)
+ 1/K : (o'=1) & (N'=16)
+ 1/K : (o'=1) & (N'=17)
+ 1/K : (o'=1) & (N'=18)
+ 1/K : (o'=1) & (N'=19)
+ 1/K : (o'=1) & (N'=20);
[mess] o=1 & ack<N -> (o'=2); // send message
[mess] o=1 & ack=N -> (o'=3); // send last message
[ack] o=2 & ack<N-2 -> (o'=1) & (ack'=min(ack+1,N)); // receive
ack & not last but one ack
[ack] o=2 & ack=N-2 -> (o'=2) & (ack'=min(ack+1,N)); // receive
ack & last but one (so now wait for ack before sending last
message)
[ack] o=2 & ack=N-1 & N>1 -> (o'=1) & (ack'=min(ack+1,N)); //
receive last ack and N>1
[ack] o=2 & ack=N-1 & N=1 -> (o'=3) & (ack'=min(ack+1,N)); //
receive last ack and N=1

[done] o=3 -> true; // finished so loop

endmodule

module recipient

  r : [0..3]; // local state of recipient
  // 0 initial state
  // 1 receive messages
  // 2 send acks
  // 3 done
  mess : [0..K]; // number of mess the originator has received

  counter : [0..1];

  [req] r=0 -> (r'=1); // send request
  [mess] r=1 & mess<N-1 -> (r'=2) & (mess'=min(mess+1,N)); //
receive message
  [mess] r=1 & mess=0 & N=1 -> (r'=2) & (mess'=min(mess+1,N)); //
receive last message and N=1

```

```

    [mess] r=1 & mess=N-1 & N>1 -> (r'=3) & (mess'=min(mess+1,N));
// receive last message and N>1
    [ack] r=2 & mess<N-1 -> (r'=1); // send ack & not last but one
    [ack] r=2 & N=1 -> (r'=3); // send last ack and N=1
    [ack] r=2 & N>1 & mess=N-1 & counter=0 -> (r'=2) & (counter'=1);
// send last but one ack and N>1
    [ack] r=2 & N>1 & mess=N-1 & counter=1 -> (r'=1) & (counter'=0);
// send last ack and N>1

    [done] r=3 -> true; // finished so loop

endmodule

rewards "orig" // contribution to KRR the originator has
    true : ack/N;
endrewards

rewards "recip" // contribution to KRO the recipient has
    true : mess/N;
endrewards

rewards "unfair_o" //in an unfair state (for originator)
    ack<mess: 1;
endrewards

rewards "unfair_r" //in an unfair state (for recipient)
    mess<ack : 1;
endrewards

rewards "steps" // number steps (time)
    [req] true : 1;
    [mess] true : 1;
    [ack] true : 1;
endrewards

```

Figure A.3.2a: Probabilistic model PRISM code (V3)

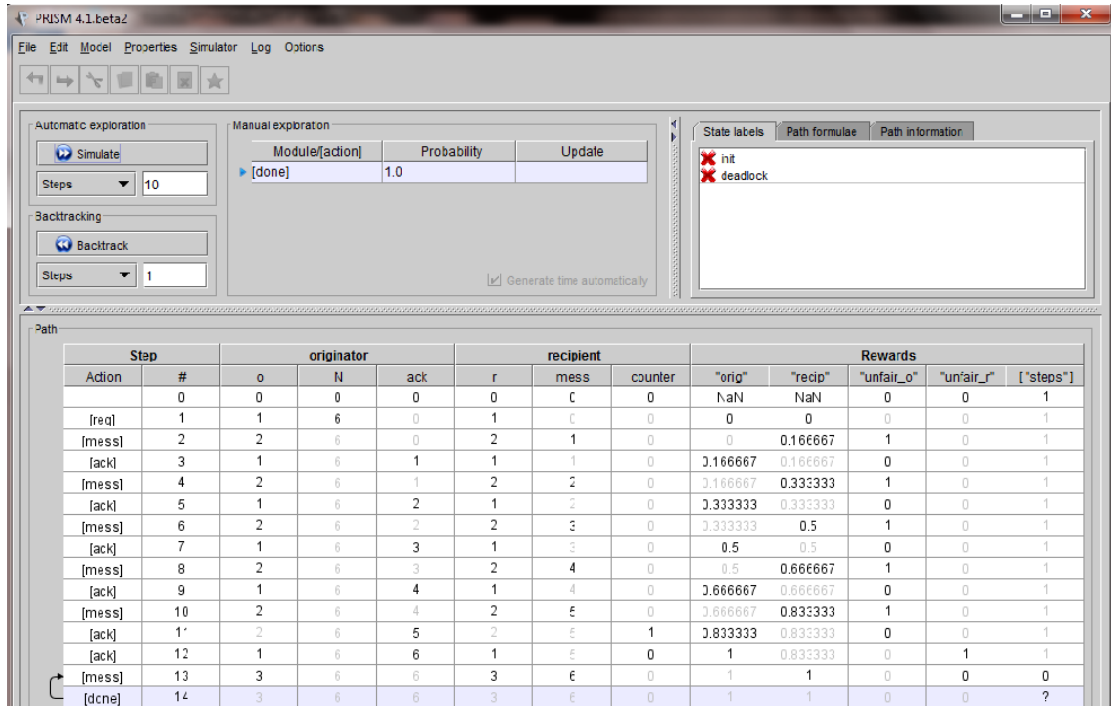


Figure A.3.2b: Probabilistic model simulation when n is randomly chosen as 6 (V3)

A.4 Version 4: PRISM codes and simulations

```
dtmc // model is a dtmc

const int N; // fix N in advance

// labels for unfairness
formula unfair_o = (ack<N & mess=N); // unfair to originator
formula unfair_r = (ack=N & mess<N); // unfair to recipient

module originator
  o : [0..3]; // local state of originator
  // 0 initial state
  // 1 send messages
  // 2 receive acks
  // 3 done
  ack : [0..N]; // number of acks the originator has received

  [req] o=0 -> (o'=1); // get request
  [mess] o=1 & ack<N -> (o'=2); // send message
  [mess] o=1 & ack=N -> (o'=3); // send last message
  [ack] o=2 & ack>1 -> (o'=1) & (ack'=min(ack+1,N)); // receive
ack & not last but one ack
  [ack] o=2 & ack=0 & N=1 -> (o'=3) & (ack'=min(ack+1,N)); //
receive last ack and N=1
  [ack] o=2 & ack=0 & N>1 -> (o'=2) & (ack'=min(ack+1,N)); //
receive ack & last but one (so now wait for ack before sending
last message)
  [ack] o=2 & ack=1 -> (o'=1) & (ack'=min(ack+1,N)); // receive
last ack and N>1
```

```

    [done] o=3 -> true; // finished so loop
endmodule

module recipient
  r : [0..3]; // local state of originator
  // 0 initial state
  // 1 receive messages
  // 2 send acks
  // 3 done
  mess : [0..N]; // number of mess the originator has received
  counter : [0..1];

  [req] r=0 -> (r'=1); // send request
  [mess] r=1 & mess<N-1 -> (r'=2) & (mess'=min(mess+1,N)); //
receive message
  [mess] r=1 & mess=0 & N=1 -> (r'=2) & (mess'=min(mess+1,N)); //
receive last message and N=1
  [mess] r=1 & mess=N-1 & N>1 -> (r'=3) & (mess'=min(mess+1,N));
// receive last message and N>1
  [ack] r=2 & mess>1 -> (r'=1); // send ack & not last but one
  [ack] r=2 & mess=1 & N=1 -> (r'=3); // send last ack and N=1
  [ack] r=2 & mess=1 & counter=0 & N>1 -> (r'=2) & (counter'=1);
// send last but one ack
  [ack] r=2 & mess=1 & counter=1 & N>1 -> (r'=1) & (counter'=0);
// send last ack (N>1)

  [done] r=3 -> true; // finished so loop
endmodule

rewards "orig" // contribution to NRR the originator has
  true : ack/N;
endrewards

rewards "recip" // contribution to NRO the recipient has
  true : mess/N;
endrewards

rewards "unfair_o" //in an unfair state (for originator)
  ack<mess : 1;
endrewards

rewards "unfair_r" //in an unfair state (for recipient)
  mess<ack : 1;
endrewards

rewards "steps" // number steps (time)
  [req] true : 1;
  [mess] true : 1;
  [ack] true : 1;
endrewards

```

Figure A.4.1a: Deterministic model PRISM codes (V4)

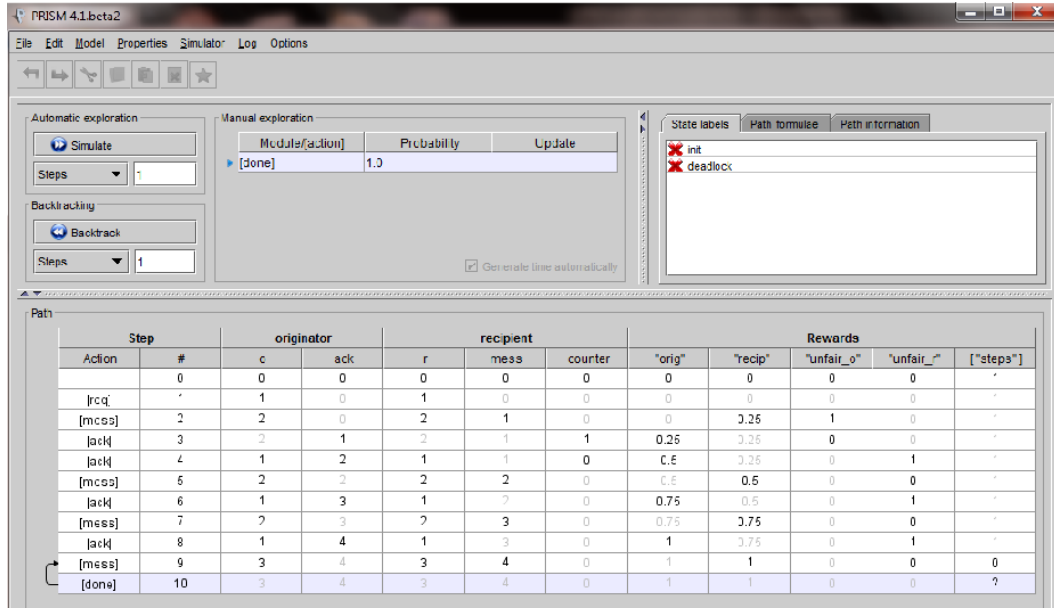


Figure A.4.1b: Deterministic model simulation when n=4 (V4)

```
dtmc // model is a dtmc
```

```
const int K; // maximum number of messages
// number of messages uniformly chosen over 1,...,K
```

```
// labels for unfairness
formula unfair_o = (ack<N & mess=N); // unfair to originator
formula unfair_r = (ack=N & mess<N); // unfair to recipient
```

```
module originator
```

```
o : [0..3]; // local state of originator
// 0 initial state
// 1 send messages
// 2 receive acks
// 3 done
```

```
N : [0..K]; // number of messages
```

```
ack : [0..K]; // number of acks the originator has received
```

```
// get request (so set K)
```

```
[req] o=0 & K=1 -> 1/K : (o'=1) & (N'=1);
```

```
[req] o=0 & K=2 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2);
```

```
[req] o=0 & K=3 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3);
```

```
[req] o=0 & K=4 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4);
```

```
[req] o=0 & K=5 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5);
```

```

[req] o=0 & K=6 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6);
[req] o=0 & K=7 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7);
[req] o=0 & K=8 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7)
      + 1/K : (o'=1) & (N'=8);
[req] o=0 & K=9 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7)
      + 1/K : (o'=1) & (N'=8)
      + 1/K : (o'=1) & (N'=9);
[req] o=0 & K=10 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7)
      + 1/K : (o'=1) & (N'=8)
      + 1/K : (o'=1) & (N'=9)
      + 1/K : (o'=1) & (N'=10);
[req] o=0 & K=11 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7)
      + 1/K : (o'=1) & (N'=8)
      + 1/K : (o'=1) & (N'=9)
      + 1/K : (o'=1) & (N'=10)
      + 1/K : (o'=1) & (N'=11);
[req] o=0 & K=12 -> 1/K : (o'=1) & (N'=1)
      + 1/K : (o'=1) & (N'=2)
      + 1/K : (o'=1) & (N'=3)
      + 1/K : (o'=1) & (N'=4)
      + 1/K : (o'=1) & (N'=5)
      + 1/K : (o'=1) & (N'=6)
      + 1/K : (o'=1) & (N'=7)

```

```

+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12);
[req] o=0 & K=13 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13);
[req] o=0 & K=14 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14);
[req] o=0 & K=15 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15);
[req] o=0 & K=16 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)

```



```

+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15)
+ 1/K : (o'=1) & (N'=16);
[req] o=0 & K=17 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15)
+ 1/K : (o'=1) & (N'=16)
+ 1/K : (o'=1) & (N'=17);
[req] o=0 & K=18 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15)
+ 1/K : (o'=1) & (N'=16)
+ 1/K : (o'=1) & (N'=17)
+ 1/K : (o'=1) & (N'=18);
[req] o=0 & K=19 -> 1/K : (o'=1) & (N'=1)
+ 1/K : (o'=1) & (N'=2)
+ 1/K : (o'=1) & (N'=3)
+ 1/K : (o'=1) & (N'=4)
+ 1/K : (o'=1) & (N'=5)
+ 1/K : (o'=1) & (N'=6)
+ 1/K : (o'=1) & (N'=7)
+ 1/K : (o'=1) & (N'=8)
+ 1/K : (o'=1) & (N'=9)
+ 1/K : (o'=1) & (N'=10)
+ 1/K : (o'=1) & (N'=11)
+ 1/K : (o'=1) & (N'=12)
+ 1/K : (o'=1) & (N'=13)
+ 1/K : (o'=1) & (N'=14)
+ 1/K : (o'=1) & (N'=15)
+ 1/K : (o'=1) & (N'=16)
+ 1/K : (o'=1) & (N'=17)
+ 1/K : (o'=1) & (N'=18)

```

```

    + 1/K : (o'=1) & (N'=19);
[req] o=0 & K=20 -> 1/K : (o'=1) & (N'=1)
    + 1/K : (o'=1) & (N'=2)
    + 1/K : (o'=1) & (N'=3)
    + 1/K : (o'=1) & (N'=4)
    + 1/K : (o'=1) & (N'=5)
    + 1/K : (o'=1) & (N'=6)
    + 1/K : (o'=1) & (N'=7)
    + 1/K : (o'=1) & (N'=8)
    + 1/K : (o'=1) & (N'=9)
    + 1/K : (o'=1) & (N'=10)
    + 1/K : (o'=1) & (N'=11)
    + 1/K : (o'=1) & (N'=12)
    + 1/K : (o'=1) & (N'=13)
    + 1/K : (o'=1) & (N'=14)
    + 1/K : (o'=1) & (N'=15)
    + 1/K : (o'=1) & (N'=16)
    + 1/K : (o'=1) & (N'=17)
    + 1/K : (o'=1) & (N'=18)
    + 1/K : (o'=1) & (N'=19)
    + 1/K : (o'=1) & (N'=20);
[mess] o=1 & ack<N -> (o'=2); // send message
[mess] o=1 & ack=N -> (o'=3); // send last message
[ack] o=2 & ack>1 -> (o'=1) & (ack'=min(ack+1,N)); // receive
ack & not last but one ack
[ack] o=2 & ack=0 & N=1 -> (o'=3) & (ack'=min(ack+1,N)); //
receive last ack and N=1
[ack] o=2 & ack=0 & N>1 -> (o'=2) & (ack'=min(ack+1,N)); //
receive ack & last but one (so now wait for ack before sending
last message)
[ack] o=2 & ack=1 -> (o'=1) & (ack'=min(ack+1,N)); // receive
last ack and N>1

[done] o=3 -> true; // finished so loop

endmodule

module recipient

    r : [0..3]; // local state of recipient
    // 0 initial state
    // 1 receive messages
    // 2 send acks
    // 3 done
    mess : [0..K]; // number of mess the originator has received

    counter : [0..1];

    [req] r=0 -> (r'=1); // send request
    [mess] r=1 & mess<N-1 -> (r'=2) & (mess'=min(mess+1,N)); //
receive message
    [mess] r=1 & mess=0 & N=1 -> (r'=2) & (mess'=min(mess+1,N)); //
receive last message and N=1
    [mess] r=1 & mess=N-1 & N>1 -> (r'=3) & (mess'=min(mess+1,N));
// receive last message and N>1
    [ack] r=2 & mess>1 -> (r'=1); // send ack & not last but one
    [ack] r=2 & mess=1 & N=1 -> (r'=3); // send last ack and N=1

```

```

    [ack] r=2 & mess=1 & counter=0 & N>1 -> (r'=2) & (counter'=1);
// send last but one ack
    [ack] r=2 & mess=1 & counter=1 & N>1 -> (r'=1) & (counter'=0);
// send last ack (N>1)

    [done] r=3 -> true; // finished so loop

endmodule

rewards "orig" // contribution to KRR the originator has
    true : ack/N;
endrewards

rewards "recip" // contribution to KRO the recipient has
    true : mess/N;
endrewards

rewards "unfair_o" //in an unfair state (for originator)
    ack<mess : 1;
endrewards

rewards "unfair_r" //in an unfair state (for recipient)
    mess<ack : 1;
endrewards

rewards "steps" // number steps (time)
    [req] true : 1;
    [mess] true : 1;
    [ack] true : 1;
endrewards

```

Figure A.4.2a: Probabilistic model PRISM code (V4)

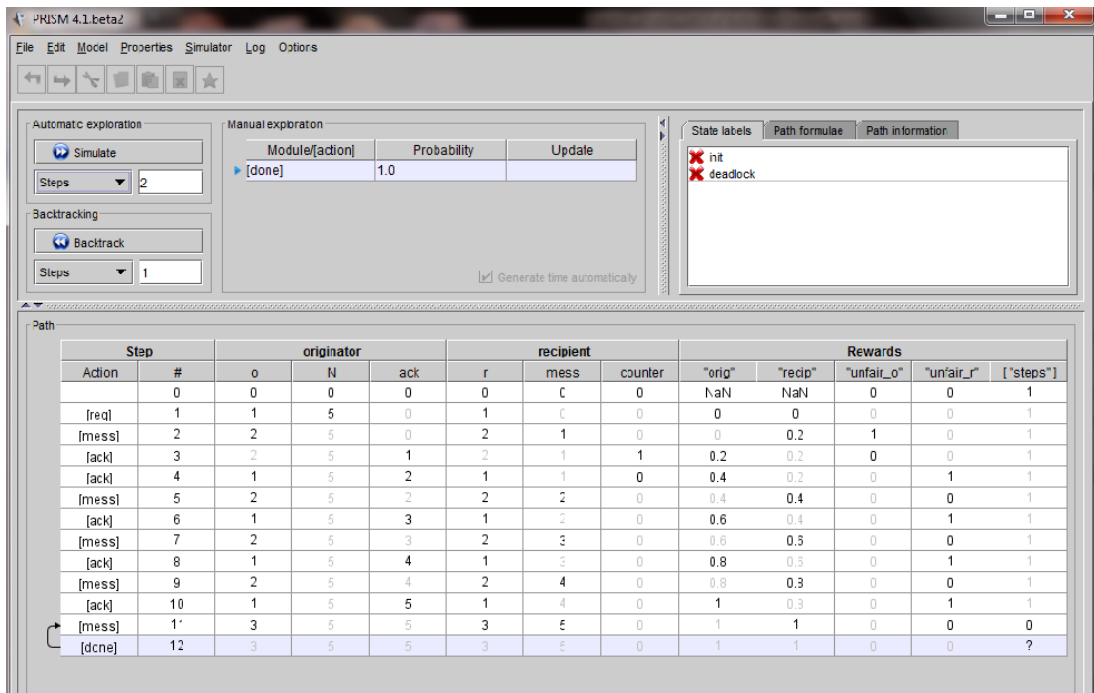


Figure A.4.2b: Probabilistic model simulation when n is randomly chosen as 5 (V4)

A.5 Version 5: PRISM codes and simulation

```

dtmc // model is a dtmc

const int N; // fix N in advance

// labels for unfairness
formula unfair_o = (ack<N & mess=N); // unfair to originator
formula unfair_r = (ack=N & mess<N); // unfair to recipient

module originator

    o : [0..3]; // local state of originator
    // 0 initial state
    // 1 send messages
    // 2 receive acks
    // 3 done
    ack : [0..N]; // number of acks the originator has received
    o_even : [0..1]; // even or odd phase (starts odd)

    [req] o=0 -> (o'=1) & (o_even'=0); // get request (start in
phase 1 which is odd)
    // odd phase
    [mess] o=1 & o_even=0 -> (o'=2); // send message
    [ack] o=2 & o_even=0 & ack<N-1 -> (o'=2) & (ack'=min(ack+1,N)) &
(o_even'=1); // receive ack & not last
    [ack] o=2 & o_even=0 & ack=N-1 -> (o'=3) & (ack'=min(ack+1,N));
// receive last ack

    // even phase
    [ack] o=2 & o_even=1 -> (o'=1) & (ack'=min(ack+1,N)); // receive
ack
    [mess] o=1 & o_even=1 & ack<N -> (o'=1) & (o_even'=0); // send
message & not last
    [mess] o=1 & o_even=1 & ack=N -> (o'=3); // send last message

    [done] o=3 -> true; // finished so loop

endmodule

module recipient

    r : [0..3]; // local state of originator
    // 0 initial state
    // 1 receive messages
    // 2 send acks
    // 3 done
    mess : [0..N]; // number of mess the originator has received
    r_even : [0..1]; // even or odd phase (starts odd)

    [req] r=0 -> (r'=1) & (r_even'=0); // send request

    // odd phase
    [mess] r=1 & r_even=0 -> (r'=2) & (mess'=min(mess+1,N)); //
receive message
    [ack] r=2 & r_even=0 & mess<N -> (r'=2) & (r_even'=1); // send
ack & not last

```

```

[ack] r=2 & r_even=0 & mess=N -> (r'=3); // send last ack

// even phase
[ack] r=2 & r_even=1 -> (r'=1); // send ack
[mess] r=1 & r_even=1 & mess<N-1 -> (r'=1) &
(mess'=min(mess+1,N)) & (r_even'=0); // receive message
[mess] r=1 & r_even=1 & mess=N-1 -> (r'=3) &
(mess'=min(mess+1,N)); // receive message and last

[done] r=3 -> true; // finished so loop

endmodule

rewards "orig" // contribution to NRR the originator has
true : ack/N;
endrewards

rewards "recip" // contribution to NRO the recipient has
true : mess/N;
endrewards

rewards "unfair_o" //in an unfair state (for originator)
ack<mess : 1;
endrewards

rewards "unfair_r" //in an unfair state (for recipient)
mess<ack : 1;
endrewards

rewards "steps" // number steps (time)
[req] true : 1;
[mess] true : 1;
[ack] true : 1;
endrewards

```

Figure A.5.1a: Deterministic model Prism codes (V5)

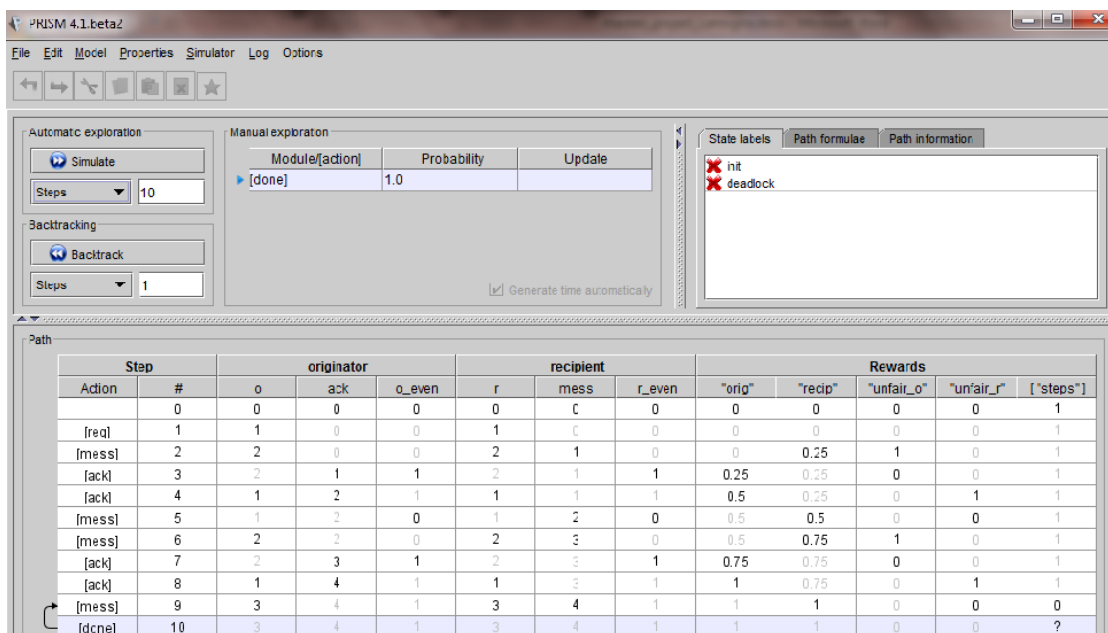


Figure A.5.1b: Deterministic model simulation when n is 4 (V5)

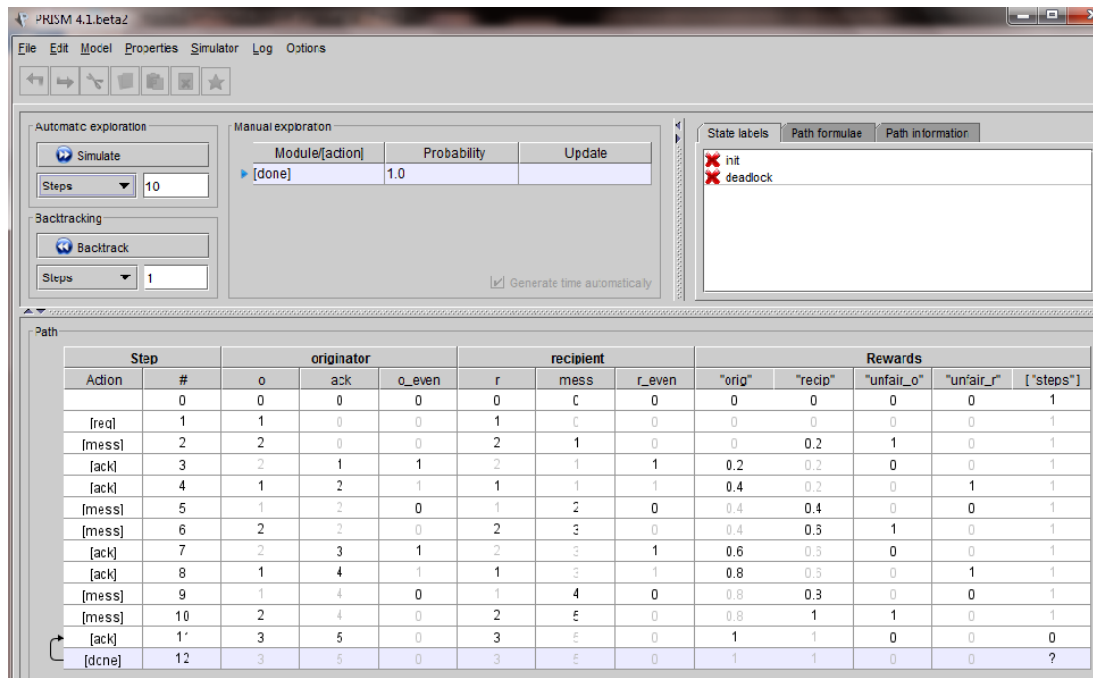


Figure A.5.1c: Deterministic simulation when n is 5 (V5).

```
dtmc // model is a dtmc
```

```
const int K; // maximum number of messages
// number of messages uniformly chosen over 1,...,K
```

```
// labels for unfairness
formula unfair_o = (ack<N & mess=N); // unfair to originator
formula unfair_r = (ack=N & mess<N); // unfair to recipient
```

```
module originator
```

```
o : [0..3]; // local state of originator
// 0 initial state
// 1 send messages
// 2 receive acks
// 3 done
```

```
N : [0..K]; // number of messages
ack : [0..K]; // number of acks the originator has received
o_even : [0..1]; // even or odd phase (starts odd)
```

```
// get request (so set K)
[req] o=0 & K=1 -> 1/K : (o'=1) & (o_even'=0) & (N'=1);
[req] o=0 & K=2 -> 1/K : (o'=1) & (o_even'=0) & (N'=1)
+ 1/K : (o'=1) & (o_even'=0) & (N'=2);
[req] o=0 & K=3 -> 1/K : (o'=1) & (o_even'=0) & (N'=1)
+ 1/K : (o'=1) & (o_even'=0) & (N'=2)
+ 1/K : (o'=1) & (o_even'=0) & (N'=3);
```

```
[req] o=0 & K=4 -> 1/K : (o'=1) & (o_even'=0) & (N'=1)
+ 1/K : (o'=1) & (o_even'=0) & (N'=2)
+ 1/K : (o'=1) & (o_even'=0) & (N'=3)
+ 1/K : (o'=1) & (o_even'=0) & (N'=4);
```

```
[req] o=0 & K=5 -> 1/K : (o'=1) & (o_even'=0) & (N'=1)
+ 1/K : (o'=1) & (o_even'=0) & (N'=2)
```

[illegible]

[illegible]

[illegible]

```

        + 1/K : (o'=1) & (o_even'=0) & (N'=16)
        + 1/K : (o'=1) & (o_even'=0) & (N'=17)
        + 1/K : (o'=1) & (o_even'=0) & (N'=18)
        + 1/K : (o'=1) & (o_even'=0) & (N'=19);
[req] o=0 & K=20 -> 1/K : (o'=1) & (o_even'=0) & (N'=1)
        + 1/K : (o'=1) & (o_even'=0) & (N'=2)
        + 1/K : (o'=1) & (o_even'=0) & (N'=3)
        + 1/K : (o'=1) & (o_even'=0) & (N'=4)
        + 1/K : (o'=1) & (o_even'=0) & (N'=5)
        + 1/K : (o'=1) & (o_even'=0) & (N'=6)
        + 1/K : (o'=1) & (o_even'=0) & (N'=7)
        + 1/K : (o'=1) & (o_even'=0) & (N'=8)
        + 1/K : (o'=1) & (o_even'=0) & (N'=9)
        + 1/K : (o'=1) & (o_even'=0) & (N'=10)
        + 1/K : (o'=1) & (o_even'=0) & (N'=11)
        + 1/K : (o'=1) & (o_even'=0) & (N'=12)
        + 1/K : (o'=1) & (o_even'=0) & (N'=13)
        + 1/K : (o'=1) & (o_even'=0) & (N'=14)
        + 1/K : (o'=1) & (o_even'=0) & (N'=15)
        + 1/K : (o'=1) & (o_even'=0) & (N'=16)
        + 1/K : (o'=1) & (o_even'=0) & (N'=17)
        + 1/K : (o'=1) & (o_even'=0) & (N'=18)
        + 1/K : (o'=1) & (o_even'=0) & (N'=19)
        + 1/K : (o'=1) & (o_even'=0) & (N'=20);
// odd phase
[mess] o=1 & o_even=0 -> (o'=2); // send message
[ack] o=2 & o_even=0 & ack<N-1 -> (o'=2) & (ack'=min(ack+1,K)) &
(o_even'=1); // receive ack & not last
[ack] o=2 & o_even=0 & ack=N-1 -> (o'=3) & (ack'=min(ack+1,K));
// receive last ack

// even phase
[ack] o=2 & o_even=1 -> (o'=1) & (ack'=min(ack+1,K)); // receive
ack
[mess] o=1 & o_even=1 & ack<N -> (o'=1) & (o_even'=0); // send
message & not last
[mess] o=1 & o_even=1 & ack=N -> (o'=3); // send last message

[done] o=3 -> true; // finished so loop

endmodule

module recipient

    r : [0..3]; // local state of recipient
    // 0 initial state
    // 1 receive messages
    // 2 send acks
    // 3 done
    mess : [0..K]; // number of mess the originator has received
    r_even : [0..1]; // even or odd phase (starts odd)
    counter : [0..1];

    [req] r=0 -> (r'=1) & (r_even'=0); // send request

    // odd phase
    [mess] r=1 & r_even=0 -> (r'=2) & (mess'=min(mess+1,K)); //
receive message

```

```

    [ack] r=2 & r_even=0 & mess<N -> (r'=2) & (r_even'=1); // send
ack & not last
    [ack] r=2 & r_even=0 & mess=N -> (r'=3); // send last ack

    // even phase
    [ack] r=2 & r_even=1 -> (r'=1); // send ack
    [mess] r=1 & r_even=1 & mess<N-1 -> (r'=1) &
(mess'=min(mess+1,K)) & (r_even'=0); // receive message
    [mess] r=1 & r_even=1 & mess=N-1 -> (r'=3) &
(mess'=min(mess+1,K)); // receive message and last

    [done] r=3 -> true; // finished so loop

endmodule

rewards "orig" // contribution to KRR the originator has
    true : ack/N ;
endrewards

rewards "recip" // contribution to KRO the recipient has
    true : mess/N;
endrewards

rewards "unfair_o" //in an unfair state (for originator)
    ack<mess : 1;
endrewards

rewards "unfair_r" //in an unfair state (for recipient)
    mess<ack : 1;
endrewards

rewards "steps" // number steps (time)
    [req] true : 1;
    [mess] true : 1;
    [ack] true : 1;
endrewards

```

Figure A.5.2a: Probabilistic model Prism code (V5)

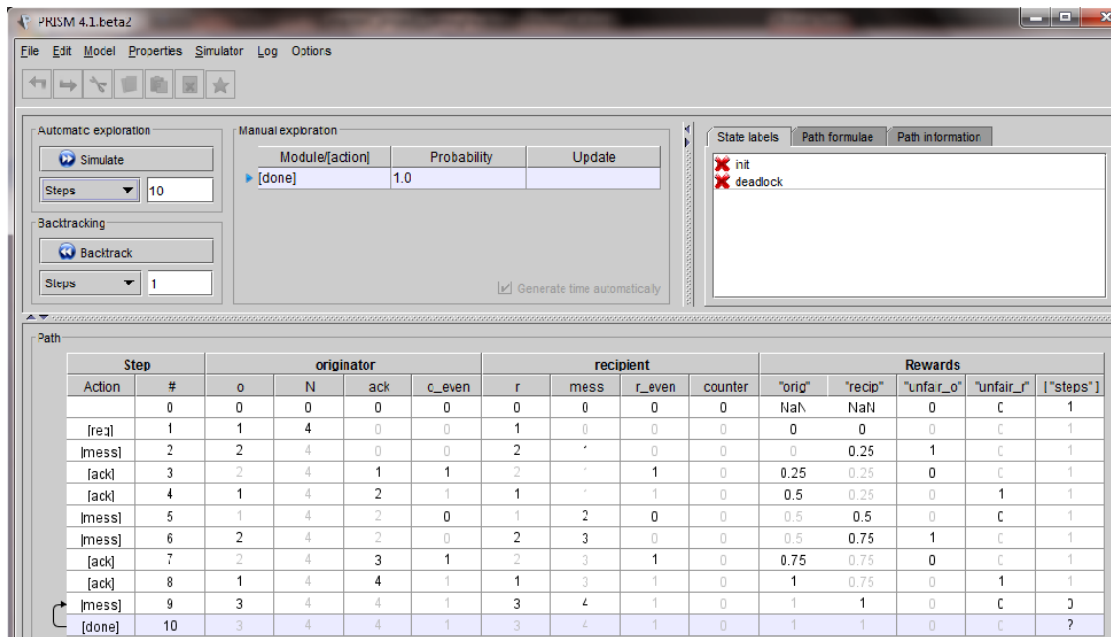


Figure A.5.2b: Probabilistic model simulation when n is randomly chosen as 4 (V5)

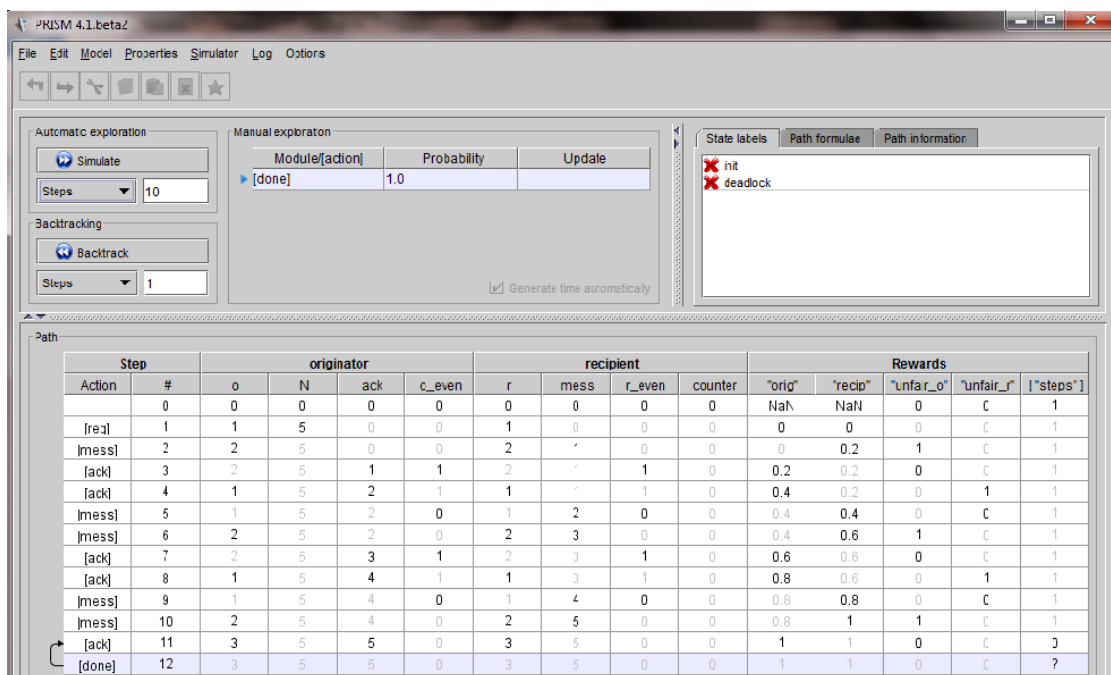


Figure A.5.2c: Probabilistic model simulation when n is randomly chosen as 5 (V5)

Bibliography

1. *Probabilistic Non-repudiation Without Trusted Third Party*. **Olivier Markowitch and Yves Roggeman**. Amalfi, Italy : In Second Conference on Security in Communication Networks 1999 , September 1999.
2. *A Fair Protocol For Signing Contracts*. **Michael Ben-or, Odded Goldreich, Silvio Micali, and Ronald L. Rivest**. 1, s.l. : IEEE Transactions on Information Theory, January 1990, Vol. 36.
3. *Game Relations, Metrics and Refinements*. **Raman, Vishwanath**. Phd thesis in University of California, Santa Cruz : s.n., 2010.
4. *Model Checking for Probabilistic Timed Automata*. **G. Norman, D. Parker and J. Sproston**. s.l. : Formal Methods in System Design, 2012.
5. prismmodelchecker. *PRISM website*. [Online] [Cited: March 21, 2013.] <http://www.prismmodelchecker.org>.
6. **Marta Kwiatkowska, Gethin Norman and Dave Parker**. *Probabilistic Model Checking lecture course at University of Oxford*. [<http://www.prismmodelchecker.org/lectures/biss07/01-intro.pdf>] 2001.
7. *PRISM-games: A Model Checker for Stochastic Multi-Player Games*. **Taolue Chen, Vojtech Forejt, Marta Kwiatkowska, David Parker and Aistis Simaitis**. Department of Computer Science, University of Oxford, UK : s.n.
8. *PRISM 4.0: Verification of Probabilistic Real-time Systems*. **Kwaitkowska, Marta, Norman, Gethin and Parker, David**. Springer : In proc. 23rd International Conferecne on Computer Aided Verification, 2011. Vol. 6806 of LNCS. CAV'11.
9. *Model Checking for Probabilistic and Nondeterministic System*. **Bianco, A. and L., Alfaro**. Springer : s.n., 1995. 15th Conference on Foundations of Software Technology and Theoretical Computer Science. Vol. 1026 of LNCS.
10. *A Logic for Reasoning About Time and Reliability*. **Hansson, H. and Jonsson, B.** 5, 1994, Formal Aspects of Computing, Vol. 6, pp. 512-535.
11. *Verifying Continuous Time Markov Chains*. **A., Aziz, et al.** [ed.] R. Alur and T. Henzinger. Springer : s.n., 1996. Proc. 8th Int. Conf. Computer Aided Verification. Vol. 1102 of LNCS.
12. *Approximate Symbolic Model Checking of Continuos-time Markov Chains*. **Baier, c., Katoen, J.-P and Hermanns, H.** [ed.] J. Baeten and S. Mauw. s.l. : Springer, 1999. Proc. 10th International Conference on concurrency Theory (CONCUR'99). Vol. 1664 of LNCS, pp. 146-161.

13. *The temporal logic of Programs*. **Pnueli, Amir**. Providence, RI, USA : s.n., 1977. Proc. of the 18th Annual Symposium on Foundations of Computer Science (FOCS). pp. 46-57. ISSN 0272-5428.
14. **Baier, C.** *On Algorithmic Verification Methods for Probabilistic Systems*. Fakultät für Mathematik & Informatik, Universität Mannheim. 1998. Habilitation thesis.
15. **Parker, Dave**. Probabilistic model checking. *Lecture 2 Discrete-time Markov Chains*. 2011.
16. —. Probabilistic Model Checking. *Lecture 3 Discrete-time Markov Chains*. 2011.
17. **Kwiatkowska, Marta, Norman, Gethin and Parker, David**. Stochastic Model Checking. June 2007, Vol. 4486 of Lecture Notes in Computer Science (Tutorial Volume), pp. 220-270.
18. *A Game-Based Verification of Non-Repudiation and Fair Exchange Protocols*. **Raskin, Steve Kremer and Jean-Francois**. pp 551-565, Belgium, Lecture Notes in Computer Science : CONCUR 2001, 2001, Vol. 2154.
19. *Parametric Probabilistic Transition Systems For system Design And Analysis*. **R. Lanotte, A. Maggiolo-Schettini and A. Troina**. s.l. : Formal Aspects of Computing, March 2007, Vols. 19(1) : 93-109.
20. *Automatic analysis of a non-repudiation protocol*. **R.Lanotte, A. Maggiolo-Schettini and A. Troina**. s.l. : Proc. Second Workshop Quantitative Aspects of Programming Languages (QAPL 2004), 2005, Vol. 112. pages 113-129.
21. *Quantitative Analysis of a Probabilistic Non-repudiation Protocol through Model Checking*. **Mukhopadhyay, I. Saha and D.** s.l. : Proc. 5th International Conference on Information Systems Security, 2009. Vol. 5905 of Lecture notes in Computer Science. pages 292-300. ICISS'09.
22. *An Optimistic Non Repudiation Protocol with Transparent Trusted Third Party*. **Markowitch, Olivier and Kremer, Steve**. 2001, Lecture Notes in computer science, Vol. 2200, pp. 363-378.
23. *Weak Bisimulation For Probabilistic Timed Automata And Applications To Security*. **R. Lanotte, A. Maggiolo-Schettini and A. Troina**. s.l. : Proc. 1st Int. Conference on Software Engineering and Formal Methods (SEFM'03), IEEE, pages 34-43, 2003.
24. *QoS Issues of Using Probabilistic Non-Repudiation Protocol in Mobile Ad Hoc Network Environment*. **Slay, Yi-Chi Lin and Jill**. South Australia : Proc. 4th Australian Information Security Management Conference, 2006.
25. *Analysis of Probabilistic Contract Signing*. **Norman, Gethin and Vitaly, Shmatikov**. 14(6), 2006, Journal of computer security, pp. 561-589.
26. **Easton, Valerie J. and McColl, John H.** STEPS Statistics Glossary. *STEPS*. [Online] 1.1, January 19, 2004. [Cited: July 28, 2013.]
http://www.stats.gla.ac.uk/steps/glossary/probability_distributions.html#unifdistn.

27. *Performance Analysis Of a Probabilistic Timed Automata Using Digital clocks.* **M.Kwaitkowska, G. Norman, D.Parker and J. Sproston.** s.l. : Springer, September 2003. In Proc. Formal Modeling and Analysis of Timed Systems (FORMATS'03). Vol. volume 2791 of LNCS, pp. 105-120.

28. *Stochastic games for verification of probabilistic timed automata.* **M. Kwiatkowska, G. Norman, D. Parker.** Springer : s.n., September 2009. In Proc. 7th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'09). Vol. 5813 of LNCS, pp. 212-227.

29. *Estimating The Maximum Information Leakage.* **Pierro, A. Aldini and A. Di-Pierro.** s.l. : International Journal of Information Security 7, 219-242, 2008.

