



The United Nations  
University

**UNU-IIST**

International Institute for  
Software Technology

---

# Formal Analysis of Streaming Down- loading Protocol for System Upgrad- ing

---

Miaomiao Zhang and Dang Van Hung

December 2005

## UNU-IIST and UNU-IIST Reports

UNU-IIST (United Nations University International Institute for Software Technology) is a Research and Training Centre of the United Nations University (UNU). It is based in Macau, and was founded in 1991. It started operations in July 1992. UNU-IIST is jointly funded by the Governor of Macau and the governments of the People's Republic of China and Portugal through a contribution to the UNU Endowment Fund. As well as providing two-thirds of the endowment fund, the Macau authorities also supply UNU-IIST with its office premises and furniture and subsidise fellow accommodation.

The mission of UNU-IIST is to assist developing countries in the application and development of software technology.

UNU-IIST contributes through its programmatic activities:

1. Advanced development projects, in which software techniques supported by tools are applied,
2. Research projects, in which new techniques for software development are investigated,
3. Curriculum development projects, in which courses of software technology for universities in developing countries are developed,
4. University development projects, which complement the curriculum development projects by aiming to strengthen all aspects of computer science teaching in universities in developing countries,
5. Schools and Courses, which typically teach advanced software development techniques,
6. Events, in which conferences and workshops are organised or supported by UNU-IIST, and
7. Dissemination, in which UNU-IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU-IIST is on formal methods for software development. UNU-IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU-IIST produces a report series. Reports are either Research **[R]**, Technical **[T]**, Compendia **[C]** or Administrative **[A]**. They are records of UNU-IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU-IIST at P.O. Box 3058, Macau or visit UNU-IIST's home page: <http://www.iist.unu.edu>, if you would like to know more about UNU-IIST and its report series.

G. M. Reed, Director



The United Nations  
University

**UNU-IIST**

International Institute for  
Software Technology

P.O. Box 3058  
Macau

---

# Formal Analysis of Streaming Downloading Protocol for System Upgrading

---

Miaomiao Zhang and Dang Van Hung

## Abstract

For a PC-mobile download system which is embedded with streaming download protocol, there are problems that the data cannot be transmitted correctly from the PC to the mobile, or the transmission is unacceptably slow. To solve these problems, we carry out a formal analysis for the protocol with some timing parameters and a given probability of losing and unordered data using a probabilistic model checking tool PRISM. We introduce a technique to reduce the state space of the system modelling the protocol which is a network of probabilistic timed automata. The experimental results in PRISM give us a clear explanation to the problems, and are helpful in identifying the optimal parameter settings to meet industrial requirements.

Zhang Miaomiao is a lecturer at the School of Software Engineering, Tongji University, Shanghai 201408, China. She holds a PhD degree from Shanghai Jiaotong University in the automation area in 2001. She becomes a UNU-IIST fellow in April 2005.

e-mail: miaomiao@iist.unu.edu

Dang Van Hung is a research fellow for the research project “Theories and Design Methods for Real-time Systems” since October 1995 being on leave of absence from Institute of Information Technology, Nghia Do, Cau Giay, Hanoi, Vietnam. He has a PhD (equivalent) degree in Computer Science (concurrent systems) in 1988, Computer and Automation Research Institute (SZTAKI), Hungarian Academy of Sciences, Budapest, Hungary, and a BSc degree in Mathematics (numerical methods) in 1977, Hanoi University, Hanoi, Vietnam.

His research interests include Formal Techniques of Programming, Concurrent and Distributed Computing, Design Techniques for Real-Time systems.

e-mail: dvh@iist.unu.edu

---

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>PC-Mobile System and Download Protocol</b>	<b>2</b>
2.1	Overview of the system . . . . .	3
2.2	System Problems . . . . .	4
<b>3</b>	<b>Abstract Probabilistic Timed Model</b>	<b>6</b>
3.1	Probabilistic Timed Automata . . . . .	6
3.2	Modelling Ideas . . . . .	8
3.3	Protocol Model . . . . .	12
<b>4</b>	<b>Prism Model and Verification</b>	<b>14</b>
4.1	Model in Prism . . . . .	14
4.2	Experimental Results . . . . .	16
<b>5</b>	<b>Conclusion</b>	<b>19</b>
<b>A</b>	<b>Code in PRISM</b>	<b>21</b>



## 1 Introduction

The streaming download protocol is used to download software into cell phones which is needed for factory testing, installing and system upgrading. Implementation of this protocol by a communication company and later used in a PC-mobile system put forward some correctness problems, such as software could not be downloaded or it takes unacceptably long time to download software. These encourage us to carry out a formal analysis for the protocol using some form of mechanical support, such as a model checker or a theorem prover. Our formal analysis is based on the real PC-mobile download system utilized by the company where this protocol is embedded. The download program transferred by this protocol is organized into packets contained within frames, and the data link between PC and mobile is two-wire USB connection.

The mechanism of this protocol is somehow similar to the sliding window protocol [10] except that here we take a different treatment for window movement and resending. Since the data link parameters including buffer size of the USB buffer and the transmission time delay are the main factors influencing the system performance, we will consider these factors carefully in our paper. Unlike in the papers in the literature [5][6] with sliding window on both the sender and the receiver, we do not consider the sliding window for the receiver (mobile) side since there are a lot of space for the memory in mobile compared with that in USB buffer. Moreover in this paper we focus on the download performance analysis while paying less attention to the processing of data frames in the mobile side.

An important expected property of this system is time related one that the program downloading could be finished as quick as possible. To measure the downloading time, we introduce timing information parameters for this system model. The typical ones are transmission time delay, timeout values and processing time in mobile side. The protocol exhibits timing and nondeterministic behaviors such as with certain amount of time elapsing a frame is transmitted or processed or lost. Besides, one key point difference between the system this paper and the one in other sliding window papers is that in this system there is likelihood of some transitions, for instance events with respect to frame lost, and unordered frame probability are taken into account. A natural model that embodies the nondeterminism, probabilistic and real-time aspects, called probabilistic timed automata which is a probabilistic extension of timed automata [1], has been proposed earlier in [2] and is adopted here to model the system.

The system we are analyzing is in the subclass of partially synchronous systems in which (i) the transmission delay of a frame via link and the processing delay in mobile side can take values nondeterministically but are bounded by some constant, and (ii) each sent frame that has not been received needs a timer to record the time elapsing since it has been sent as a base to decide whether or not to resend this frame, (iii) all the clocks used in the system evolves simultaneously, (iv) for a download software package, the number of frames is normally high, hence the number of the system states is big. Verification of these systems by model checking is often very difficult since the state space of a timed automaton grows exponentially in the number of timers. Many realistic algorithms and protocols fall into the class of "difficult" partially synchronous systems.

Examples include the sliding window protocol for the reliable transmission of data over unreliable channels [5], and the ZeroConf protocol [14][7][16] whose purpose is to dynamically configure IPv4 link-local addresses.

In the real download system, each frame of the downloaded software has a unique address to differentiate with others, while for modelling this could lead to the explosion. To avoid big state space, there is no need to give different labels to each frame of the download software. We only label those frames that currently are within current window. This suffices to guarantee the unique identification for each frame in this window. The position of the current window in the list of frames indicates the number of frames having been downloaded successfully. When the window moves, some frames at the left of the windows become out of the windows, and the same number of new frames come to the right of the windows. The labels of the frames which leave the windows are then reused for the newly coming frames of the window. So, a label is always unique among the sent frames that have not been removed from the window, and hence cause no problem for the protocol mechanism though it may have been used for frames that have been downloaded successfully. Since each sent frame corresponds to a timer, the timer assignment and released is also need to be carefully considered in this paper according to this label setting.

In this paper, we focus on the qualitative behavior of the system similarly to the performance analysis of a sliding window protocol by the simulation tool POOSL presented in the paper [3]. However, the analysis in this paper is more formal. Moreover, the algorithm, the behaviors and the mechanism used in our system are more complete and closer to the reality. Our main contribution of this paper is to explore the above proposed to model the system formally, and to modify the protocol specification by introducing some timing parameters to meet the system requirement. Experimental results with the probabilistic model checker PRISM (for the details of PRISM see [8] and [9]) with the change for the values of different parameters show the qualitative property, which can give us a clear explanation to the problems met in industry, and are helpful to identify optimal parameters settings and to determine a number of key parameters such as the timeout period, transmission time delay, processing time to improve system performance.

The rest of the paper is organized as follows. An overview of the download protocol is illustrated in the next section, together with the problems for the protocol met in practice. In Section 3, the concept of probabilistic timed automata is first recalled, followed by the analysis and construction of the abstract system model for the download protocol with a network of probabilistic timed automata. In Section 4, we use PRISM to analyze the performance of the system with respect to different values for the system parameters. Finally in Section 5, we conclude the paper.

## 2 PC-Mobile System and Download Protocol

In this section, we give an overview of the download protocol and its problems arising from industry field.

## 2.1 Overview of the system

Information transferred by this protocol is organized into packets contained within frames. A frame is a packet which has been encoded for transmission over a particular link. It has explicit minimum and maximum lengths and a set of required pieces of information that must appear within it. The link between the PC and the mobile is a two-wire usb (universal serial bus) which is a type of plug-in connection that is used to connect devices for PC and mobile, see Fig. 1, where we use channel 1 and channel 2 to stand for the two wires. So there are four main components in the system, PC, mobile, channel 1 and channel 2. Meanwhile, there are four usb buffers,  $b_1$ ,  $b_2$ ,  $b_3$  and  $b_4$  where  $b_1$  is usb sending buffer,  $b_2$  is mobile usb receiving buffer,  $b_3$  is mobile usb sending buffer and  $b_4$  is PC usb receiving buffer. Buffers  $b_1$  and  $b_2$  store frames,  $b_3$  and  $b_4$  store acknowledgment. At a time, a frame is first sent to  $b_1$ , then channel 1 transmits it from  $b_1$  to  $b_2$ . After processing the frame in the mobile side, the mobile generates a corresponding acknowledgment to  $b_3$ , then channel 2 transmits the acknowledgment to  $b_4$  so that the PC can issue other actions from the mobile response. Usb is also faster than older ports, such as serial and parallel ports. The usb 1.1 specification supports data transfer rates of up to 12Mb/sec and usb 2.0 has a maximum transfer rate of 480 Mbps. The connection is usually error-free, but occasionally there are bad connections which generate frame lost and unordered frame. We use  $p_1$  to denote the probability of message lost via the channels, and  $p_2$  to denote probability of unordered frame found in mobile side.

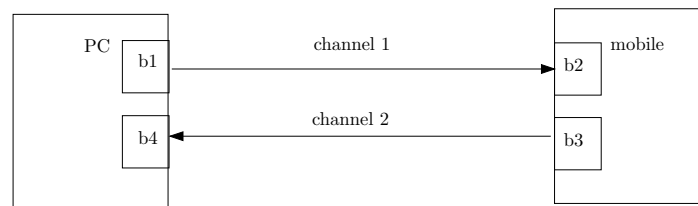


Figure 1: Downloading system

To download software from PC, first the PC sends a *hello* frame through the wire channel 1 to the mobile to inform it will start to transfer software and at the same time starts a timer  $z$ . If the mobile is ready for accepting the data, it will send an *AckH* back to the PC through the wire *channel2*. Otherwise, the mobile sends nothing. In the case that the mobile is not ready for receiving data, or the lost of the message *hello* or *AckH*, the timer  $z$  will give a timeout, which means that the PC has been waiting a certain length of time for the acknowledgement after sending a frame without receiving it. Then the PC needs to retransmit the *hello* frame again. After receiving *AckH*, the PC starts to send the real frame  $i$  of the software and starts a new timer  $t[i]$ . Each frame contains the relative address of the ROM in the mobile to which the frame will be written, in this way different frames can be distinguished by different target address of ROM in the mobile. In our later model, we will use different integers to denote the different frames. Upon receipt of the frame  $i$ , the mobile detects errors in the frame if any and discard any invalid (including bad or duplicated) frames. Errors are detected by examining the frame for valid contents and by checking the Frame Check Sequence (FCS). Successful receipt

of the frame  $i$  is responded to with an acknowledgement  $ack[i]$  which is sent back to the PC including the relative address (label) of the frame  $i$ . In the case the timer  $t[i]$  times out and  $ack[i]$  is not received, the PC needs to retransmit the frame  $i$ .

To improve the download efficiency, at any time, the sender maintains a sending sliding window of consecutive labels (relative addresses) corresponding to frames it is sending or going to send. These frames are said to be a part of the sending window. This technique allows data to be sent in one direction between a pair of protocol entities, subject to a maximum number of unacknowledged messages. The window size  $WS$  might be the maximal number of unacknowledged frames. This means that the PC will not wait for an acknowledgement frame before sending another frame in the window, provided that the cursor have not reached the right end of the window. Note here if all the frames in the window are sent, the PC will wait until the acknowledgement for the leftmost frame is received. If the PC receives an acknowledgement, it may assume that any unacknowledged frames sent before the one being acknowledged were either lost or failed and need to send them again. This implies that if the acknowledgement does not correspond to the leftmost one, the window will not move. For instance in Figure 2, the size of the sliding window is 6 and all the frames in the windows have been sent but no acknowledgement for any of them has been received. At some time, if the acknowledgement for the frame in the left position (labelled with 0) is received, the window moves right, so new frame labelled with 6 can be sent. However, if the first arriving acknowledgment corresponds to the frame 2, since all the channels keep FIFO rule, we can assume the frame 0 and frame 1 are lost and we need to send the frame 0 and the frame 1 again. Such process continues until the entire program has been transferred.

For the mobile site, after it get a frame from  $b2$ , it will get and process it, and generate an acknowledgement which will be sent back to PC if the frame is correct. The successfully received frame from PC side are temporarily stored in the RAM of the mobile for checking, if it is good. and then put into the ROM afterward after every fixed time interval according to the address within the frames.

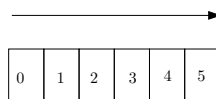


Figure 2: Sliding window

## 2.2 System Problems

The purpose of this protocol is to successfully download software to mobile for system upgrading. However during the implementation by a communication company, they encountered several problems as follows:

- The download process does not proceed, the software is not downloaded at all.

- The software is downloaded, but at a low speed. The question is how to change protocol parameters to speed up the downloading.

After carefully checking, they found that due to the very limited buffer size in  $b2$  and the fast sending frames from PC, in fact most of the frames are lost due to the slow data processing of the mobile that lead to the  $b2$  buffer overflowing. To solve the problem, in the protocol implementation they add another parameter  $TD$  as the time delay between sending two consecutive frames and hereby solve the first problem. However, the  $TD$  they set is big and it took several minutes to download a small program. Even though, they have not yet proved the solution in a formal way and is still not clear if  $TD$  they set is rational. Moreover, there are a number of parameters in the protocol, questions raised as how the parameters affect the performance and how to make the downloading as fast as possible need to be further investigated. And note here, compared with the USB buffer size there are enough RAM memory in mobile site, and only after checking there is empty place in buffer  $b3$  can the acknowledgement arrives at it, therefore there is no buffer overflowing scenario in  $b3$ .

To analyze the protocol, we introduce some time-related parameters that will appear in our later model apart from  $TO$  appearing in the protocol specification. These parameters listed as follows.

- $TD$ , as said before, is the time delay between sending two consecutively frames.
- $TO$  is the time out value, from the time point PC sending a frame, until the time point to resend the frame if no acknowledgment for this frame has been received.
- $TR$  is the maximal transmission time for a frame or acknowledgment though the channel 1 or the channel 2.
- $TP$  is the maximal processing time of a frame by the mobile, the purpose for introducing this one is for analyzing the first problem mentioned above. If the precessing of a frame is much faster than sending a frame to the channel  $b2$ , obviously  $b2$ 's overflowing can not happen.

The size  $BS$  of buffers is also a key parameter to influence protocol performance. Here we assume that the buffers  $b1$ ,  $b2$  and  $b3$  (except  $b4$ ) have the same size  $BS$ . There are also other parameters such as the number of frames to be sent  $N$  (i.e. the size of software for downloading) and the size  $WS$  of the sliding window.

The complete download system consists of not only timing constraints, but also stochastic information. For modeling a such kind of systems, we use probabilistic timed automata, which we considered as best suitable to our analysis problem.

### 3 Abstract Probabilistic Timed Model

#### 3.1 Probabilistic Timed Automata

In this section, we recall the concepts of probabilistic timed automata model and probabilistic timed structure as its semantics from [2].

**Probability distributions and Markov decision processes** A probability distribution over a set  $S$  is a mapping  $p : S \rightarrow [0, 1]$  such that the set  $\{s \mid s \in S \text{ and } p(s) > 0\}$  is finite, and  $\sum_{s \in S} p(s) = 1$ . The set of all probability distributions over  $S$  is denoted by  $\mu(S)$ .

A Markov decision procedure is a tuple  $(\mathcal{Q}, Steps)$ , where  $\mathcal{Q}$  is a set of states, and  $Steps : \mathcal{Q} \rightarrow 2^{\mu(\mathcal{Q})}$  is a function assigning a set of probability distributions to each state. The intuition is that the Markov decision process traverses the state space by making transitions determined by  $Steps$ : in a state  $s$ , the process selects nondeterministically a probability distribution  $p$  in  $Steps(s)$ , and then makes a probabilistic choice according to  $p$  as to which state to move to. As in [2] we label the action selecting a probability distribution with a letter from  $\Sigma$ , and assume that  $Steps : \mathcal{Q} \rightarrow 2^{\Sigma \times \mu(\mathcal{Q})}$ . So, a transition is of the form  $q \xrightarrow{a,p} q'$ , where  $(a,p) \in \Sigma \times \mu(\mathcal{Q})$  is the label of the transition. We also assume a labeling function  $L : \mathcal{Q} \rightarrow 2^{AP}$ , where  $AP$  is a set of atomic propositions, that associates a state  $q$  with the set of atomic propositions that hold at state  $q$ . Then, a labeled Markov decision process is a tuple  $(\mathcal{Q}, Steps, L)$ .

Labeled paths (or execution sequences) are nonempty finite or infinite sequence of consecutive transitions the form

$$\omega = q_0 \xrightarrow{l_0} q_1 \xrightarrow{l_1} q_2 \xrightarrow{l_2} \dots,$$

where  $q_i$  are states and  $l_i$  are labels for transitions. For a path  $\omega$ , let  $first(\omega)$  denote the first state of  $\omega$ , and if  $\omega$  is finite then let  $last(\omega)$  denote the last state of  $\omega$ .  $|\omega|$  is the length of  $\omega$  and is defined as the number of transition occurrences in  $\omega$  which is  $\infty$  if  $\omega$  is infinite. For  $k \leq |\omega|$ , let  $\omega(k)$  denote the  $k$ th state of  $\omega$ , and  $step(\omega, k)$  denote the label of the  $k$ th transition in  $\omega$ . For two paths  $\omega = q_0 \xrightarrow{l_0} q_1 \xrightarrow{l_1} q_2 \xrightarrow{l_2} \dots q_n$  and  $\omega' = q'_0 \xrightarrow{l'_0} q'_1 \xrightarrow{l'_1} q'_2 \xrightarrow{l'_2} \dots$  such that  $q_n = q'_0$ , the concatenation of  $\omega$  and  $\omega'$  is defined as  $\omega\omega' = q_0 \xrightarrow{l_0} q_1 \xrightarrow{l_1} q_2 \xrightarrow{l_2} \dots q_n \xrightarrow{l'_0} q'_1 \xrightarrow{l'_1} q'_2 \xrightarrow{l'_2} \dots$

**Clocks, clock valuations, clock constraints:** Let  $\mathbb{R}$  denote the set of non negative real numbers. A clock is a real-valued variable which increase at the same rate as real time. Let  $\mathcal{C} = \{x_1, \dots, x_n\}$  be a set of clocks. A clock valuation is a function  $\nu : \mathcal{C} \rightarrow \mathbb{R}$  that assigns a real value to each clock. Let  $\mathbb{R}^{\mathcal{C}}$  denote the set of all clock valuations, and  $\mathbf{0}$  denote the clock valuation that assigns 0 to each clock in  $\mathcal{C}$ . For a set of clock  $X \subseteq \mathcal{C}$  we denote by  $\nu[X := 0]$  the

clock valuation that assigns 0 to all clock in  $X$  and agrees with  $\nu$  on all other clocks. For  $t \in \mathbb{R}$ , we write  $\nu + t$  for the clock valuation that assigns  $\nu(x) + t$  to each clock  $x \in \mathcal{C}$ .

A constraint over  $\mathcal{C}$  is an expression of the form  $x_i \sim c$  or  $x_i - x_j \sim c$ , where  $i \neq j$ ,  $i, j \leq n$  and  $\sim \in \{<, \leq, >, \geq\}$  and  $c \in \mathbb{N}$ . A clock valuation  $\nu$  satisfies a clock constraint  $x_i \sim c$  ( $x_i - x_j \sim c$ ) iff  $\nu(x_i) \sim c$  ( $\nu(x_i) - \nu(x_j) \sim c$ ). A zone of  $\mathcal{C}$  is a convex subset of the valuation space  $\mathbb{R}^{\mathcal{C}}$  described by a conjunction of constraints. For a zone  $\zeta$  and a set of clocks  $X \subseteq \mathcal{C}$  the set  $\{\nu[X := 0] \mid \nu \in \zeta\}$  is also a zone, and is denoted by  $\zeta[X := 0]$ . Let  $\mathbf{Z}_{\mathcal{C}}$  denote the set of all zones of  $\mathcal{C}$ .

**Probabilistic Timed Automata** Timed automata were introduced in [1] as a model of real-time systems. They are extended with discrete probability distribution to model probabilistic real-time systems.

**Definition 1** *A probabilistic timed automaton is a tuple  $G = (\mathcal{S}, \mathcal{L}, \bar{s}, \mathcal{C}, inv, prob, \langle \tau_s \rangle_{s \in \mathcal{S}})$  consisting of*

- a finite set  $\mathcal{S}$  of nodes,
- a function  $\mathcal{L} : \mathcal{S} \rightarrow 2^{AP}$  assigning to each node of the automaton the set of atomic propositions that are true in that node,
- a start node  $\bar{s} \in \mathcal{S}$ ,
- a finite set  $\mathcal{C}$  of clocks,
- a function  $inv : \mathcal{S} \rightarrow \mathbf{Z}_{\mathcal{C}}$  assigning to each node an invariant condition,
- a function  $prob : \mathcal{S} \rightarrow 2^{\mu(\mathcal{S} \times 2^{\mathcal{C}})}$  assigning to each node a set of discrete probability distributions on  $\mathcal{S} \times 2^{\mathcal{C}}$ ,
- a family of functions  $\langle \tau_s \rangle_{s \in \mathcal{S}}$  where, for any  $s \in \mathcal{S}$ ,  $\tau_s : prob(s) \rightarrow \mathbf{Z}_{\mathcal{C}}$  assigns to each  $p \in prob(s)$  an enabling condition.

The last item in the definition says that all the probabilistic choices according to a probabilistic distribution (selected at a node) have the same enabling condition. The probabilistic timed automaton behaves nearly in the same way as a timed automaton does, except that it has to select a probability distribution at each discrete step. The system starts in node  $\bar{s}$  with all clocks initialized to 0. The value of the clocks increase at the same rate. At any point of time, if the system is in a node  $s$ , then the invariant  $inv(s)$  is satisfied. Then the system can either (a) remain in node  $s$  and let time advance if  $inv(s)$  is still satisfied, or (b) make a discrete transition if there exists a probability distribution  $p \in prob(s)$  such that  $\tau_s(p)$  is satisfied by the current value of the clocks. So, if letting time advance violates  $inv(s)$  then if no discrete transition could be made, the system is deadlocked.

We denote by  $\mathbf{Z}_C(G)$  the set of all clock zones occurring in  $G$ ,

$$\mathbf{Z}_C(G) = \{inv(s) \in \mathbf{Z}_C \mid s \in \mathcal{S}\} \cup \{\tau_s(p) \in \mathbf{Z}_C \mid s \in \mathcal{S} \text{ and } p \in prob(s)\}.$$

### 3.2 Modelling Ideas

In this section, we present our ideas and methods for modeling the system, mainly about how to avoid big state space and about the processing of related window move. Firstly we present some assumptions of the system for our later analysis with the PRISM tool.

1. Frames sent from the PC to the mobile arrive in the same order as the one in which they were sent, the similar assumption is for the acknowledgments sent from the mobile to the PC.
2. There is a nonzero time delay for the message transmission. This means that when the sender sends a frame, the frame can not arrive at the mobile before at least some time has elapsed.
3. Since mobile need to check whether a frame is bad or not, we assume it takes nonzero time for message processing.
4. As soon as the acknowledgment arrives at the  $b4$  buffer, it will be processed at once, and compared with the processing time in mobile, the processing time in PC is so small and can be assumed to be zero. So the buffer  $b4$  never store more than 1 message, and hence, is considered from now on in our later model as a variable.

**State space reduction and window move** In this protocol the number of frames of a downloading software could be ten thousand or more which are distinguished by different address and can be labeled by integers. For each frame, we need several various auxiliary variables to store some information related to it such as if the frame is sent or not, and if sent which clock records the time passage for its acknowledgment, or if the acknowledgment of this frame has been received or not, or which position in the sliding window the frame is located. So even for a small-size software package, the state space could be very big because of the various frames and their related variables. In order to avoid the state space exposition problem in model checking, we do not use variables to differentiate each frame of the software while only the frames in the sliding sending window are need to be kept. Suppose the sliding window can hold  $WS$  frames and position in the window is expressed by variable  $i$  ranging from 0 to  $WS - 1$ . We use the variable  $real[i]$  to denote the label of the frame in position  $i$  of the current window. The variable  $sent[i]$  is to used to indicate the frame at the position  $i$  of the window, i.e. the frame with the label  $real[i]$  has been sent or not. Similarly, the variable  $ack[i]$  indicates if the acknowledgment of the frame  $i$  with the label  $real[i]$  has been received or not by PC.

As soon as PC get a response from mobile, it will check in which position of the window, the frame label equals the acknowledgment label. More specifically, assume the position is  $j$ , i.e. the label is the same as  $real[j]$ .

- If  $j \neq 0$ , just set  $ack[j]$  as 1. Window does not move, variables regarding window movement does not change neither.
- If  $j = 0$  and starting from position 1 continuously there exists positions whose correspondent acknowledgment have been received. In other words, there exists  $m$ ,  $0 \leq m < WS - 1$ , such that  $ack[0] = 1 \wedge ack[1] = 1 \dots \wedge ack[m] = 1$ . In this case window moves  $m + 1$  steps and the position of the remaining frames in the window is decreased by  $m + 1$ , while the frame labels in  $real[i]$  starting from previous position 0 until position  $m + 1$  are moved to the place with current starting position as  $WS - (m + 1)$  and end position as  $WS - 1$  to be reused for the frames just entered the window. The related variables for the new entered frames  $ack[i]$  and  $sent[i]$ ,  $i \in \{WS - (m + 1), \dots, WS - 1\}$  are set to 0 meaning that these new frame need to be sent. We use variable  $move$  to characterize the window movement steps upon receiving an acknowledgment, and in this case,  $move = m + 1$ . Figure (3) gives the description for this case with  $WS = 6$  and  $m = 2$ .

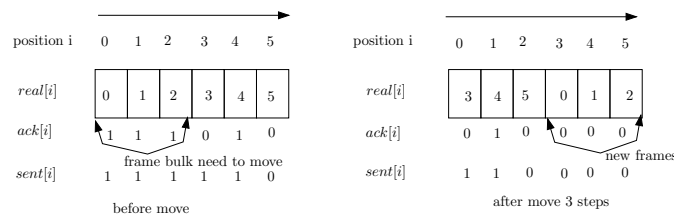


Figure 3: Window move 3 steps

Steps of window movement is accumulated by variable  $move$ , i.e.  $move$  is the cursor position of the current sliding window. If  $move$  equals the number of frames  $N$ , then all the frames are successfully arrived at mobile and download process ends.

Let the size of the current window be  $WSD$ , which is initialized to  $WS$ . When  $move$  equals  $N - WSD$ , i.e. the number of the remain frames that have not been transferred is equal to the size of the current window, we decrease the window size to  $WSD - 1$ .

**Timer evaluation** Only after a frame is sent, a timer starts up to record the time passage to judge whether or not the passing time since sending the frame expired  $TO$ . As the acknowledgment is received or timer time out, the timer is released in order to be assigned to other frames which will be sent soon. Since there are maximal  $WS$  frames in the sliding window, we need  $WS$  timers. Here we give some explanation on scheduling these timers.

We interpret the timer  $t[real[i]]$  related to the frame  $real[i]$  in the window as a clock in our model. At any time,  $t[real[i]]$  indicates the current time of the  $real[i] - th$  clock. After acknowledgment for frame  $i$  is received within  $TO$  time units, or when  $t[real[i]]$  expires after  $TO$  time units in which case we need to resend frame  $i$ ,  $t[real[i]]$  is reset to 0. Figure (4) gives the  $t[real[i]]$  evaluation before and after moving the window 3 steps, in this case it should be that acknowledgment for the frame in position 0 is just received, so  $t[3]$  is reset to 0.

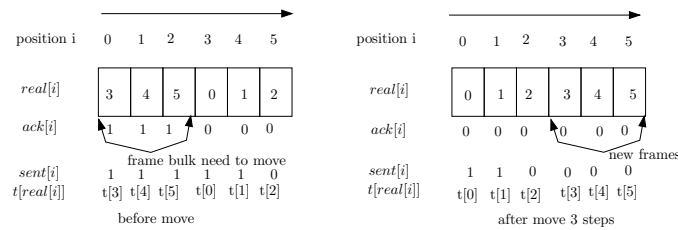


Figure 4: Timer before and after window move 3 step

If the PC receives the acknowledgment for a frame, it may assume that any unacknowledged frames sent before this one were either lost or failed and need to be sent again, and all the relevant clocks are reset to 0.

**Resend a frame** From the above mechanism, we conclude there are two cases for a frame to be resent.

- As a frame is sent, a timer is set up to record the time elapse. When the timer expires after  $TO$  time units without the acknowledgment for this frame received, then the frame need to be sent again.
- As a frame  $i$  is sent, a timer is set up to record the time passage. When an acknowledgment for a frame  $j$  sent after frame  $i$  is received, whereas acknowledgment for frame  $i$  is not received, by the assumption 1, the frame  $i$  or its acknowledgment is lost, which requires to be resent.

**Unique expression for a frame** We assume any frame and its corresponding acknowledgment have the same labels. For a frame in the window position  $j$ , by using the pair  $(move, real[j])$ , we can see the frame is uniquely expressed and is independent from implementation.

We give a summary of the procedure for the PC in Pseduo-Algol as follows.

```
data array frames[0..n-1];
integer i,j,k,move, array real[0..WS-1];
/*move is the window movement accumulation*/
```

```

timer z,td,array t[0..WS-1];
/* timer td becomes timeout after TD time units from it's start, */
/* timer t[i] becomes timeout after TO time units from it's start */
boolean array ack[0..WS-1], sent[0..WS-1];
/* initialization with hello message */
move := 0; sent:= false; ack:=false; /* windows is at the beginning,
nothing in the window has been sent nor received */
WSD:= min(WS,n); /* the data could be smaller than window */
for j=0 to WSD do real[j]:=j;
i:=0;
init: send "hello" frame to channel 1; start timer z;
await (z=timeout) or AckH received;
if not (AckH received) then goto init;
/* the communication established */
start timer td;
while move <= n - 1 do
  begin
    /* three following parts can be executed in paralell */
    /*part 1 in the loop: not atomic, and takes time */
    for i= 0 to WSD do
      /* sending frames in the window */
      if (sent[i] = false) then /* send frame i */
        begin
          await (td = timemout);
          send frames[move+i]; /*frames[move+i] has the label as real[i]*/
          start timer t[real[i]];
          start timer td; sent[i]=true
        end;

      /* PART 2 in the loop, should be performed as atomic command */
      /* check for frames that have been acknowledged */
      /* max is the index of the rightmost frame in the window
      acknowledged */
      max:=0;
      /*received_ack returns the label of the frame in the window that has been
      acknowledged */
      for i=0 to WSD do if let (v= received_ack) in (v=real[i]) then begin ack[i]:=true;
        max:=i end;
      /* resend those frame for which no ack and timed out */
      for i=0 to WSD do if (t[real[i]]=timeout and not ack[i] and sent[i])
        then sent[i]:= false;
      /* resend those frame with the index in the window lower then max
      and has not been acknowledged */
      for i=0 to max-1 do if (sent[i]=1 and not ack[i]) then
        sent[i]:=false;

      /*part 3 in the loop: should be performed as atomic command */
      /* check and move the window if necessary */
      while (ack[0] and sent[0] and move < n - WSD)
        do /* moving window one frame to the right */
          begin k:= real[0];
            for i=0 to WSD-2 do
              begin real[i]:= real[i+1];
                ack[i]:= ack[i+1]; sent[i]:= sent[i+1]
              end;
            real[WSD-1]:= k; sent[WSD-1]:= false;
            ack[WSD-1]:=false; move:= move+1;
          end
      while (ack[0] and sent[0] and move > n - WSD)
        do /* moving window one frame to the right */
          for i=0 to WSD-2 do
            begin real[i]:= real[i+1];
              ack[i]:= ack[i+1]; sent[i]:= sent[i+1]
            end;

```

variable	meaning	range
$use[i]$	equals 1 when clock $i$ is used (0 otherwise) where index $i$ corresponds to frame label $i$	0...1
$sent[i]$	equals 1 when a frame in position $i$ of the sending window has been sent (0 otherwise)	0...1
$ack[i]$	equals 1 when acknowledgment for the frame in position $i$ of the window is received (0 otherwise)	0...1
$real[j]$	frame label in position $j$ of the window	0... $WS - 1$
$t[i]$	current time since sending of frame $i$	0...TO
$WSD$	window size equals $WS$ initially	0... $WS$
$move$	move steps accumulation of the window upon receiving an acknowledgment	0... $N$
$b1[i], b2[i], b3[i]$	$i^{th}$ place respectively in buffers $b1, b2$ and $b3$ , $b1, b2$ and $b3$ have array size $BS$	0... $WS + 1$
$b4$	variable store acknowledgment in PC side	0... $WS + 1$
$nb1, nb2, nb3$	the number of places used in buffer $b1, b2, b3$ .	0... $BS$
$full$	buffer $b2$ overflows or not.	0...1
$tel$	temporary variable.	0... $WS - 1$

Table 1: Variables used in the model

```

    move:= move+1; WSD:= WSD-1;
end
end

```

### 3.3 Protocol Model

In the following, we describe the PC-mobile download system consisting of four probabilistic timed automata: PC, mobile, channel 1 and channel 2.

In addition to the integral constants mentioned above, we introduce the probability  $p1$  for the lost message rate, the probability  $p2$  for the rate of unordered frames. As window size may change when the download process nearly finishes, we therefore use the variable  $WSD$  to characterize its change. Table 1 shows the variables used in our model.

We use  $sent[i]$  to denote frame in position  $i$  of the sending window is sent or not. This includes the phenomenon both for send the frame for the first time and resend it no matter how many times it has been sent already. The value of any element of buffer arrays  $b1, b2, b3$  and variable  $b4$  range between 0 and  $WS + 1$ , where values from 0 to  $WS - 1$  indicate frame or acknowledgment stored are related with those frames in the sliding window, value  $WS$  indicates there is no frame or acknowledgment stored in this place and is therefore empty, while value  $WS + 1$  indicates the frame or acknowledgment stored is related with *hello* frame. All the elements of these buffer arrays and  $b4$  are initially set to  $WS$ , and all the other variables are set to 0.

The model for PC is shown in figure 5. In the model we also take the consideration of this case

when the time delay set between sending two consecutive frames is zero, there could happen that several timers time out at the same time, therefore each of the timers should be processed accordingly. While for the case the delay is not zero, according to model assumption this phenomenon does not arise.

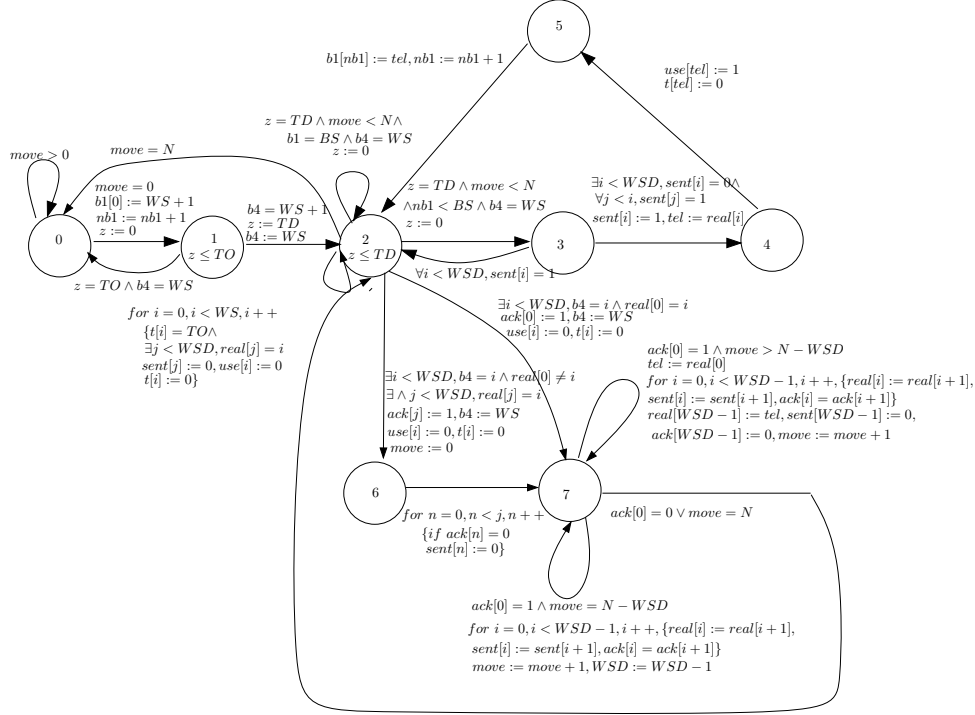


Figure 5: PC probabilistic timed automata

The model for channel 1 is shown in figure (6-a). The transition surrounding location 0 says that if there is no frames in  $b1$ , just wait until frame comes to  $b1$  ( $b1[0] \neq WS$ ) and moves to location 1. The upper bound for the transmission time delay through channel 1 is  $TR$ , according to model assumption only  $z1$  is greater than 0 can outgoing transitions from location 2 take place. When  $b2$  is full, the frame coming from  $b1$  is lost due to the  $b2$  buffer overflowing, otherwise frame successfully arrived at  $b2$ .

The model for mobile and channel 2 are shown in figure (6-b) and (7) respectively. Take Note the invariant in mobile automata is  $z2 \leq TP + TR$ . This is because even though the maximal frame processing time is  $TP$ , but in the case if  $b3$  is full then acknowledgement will still stay in RAM of mobile and can not reach  $b3$  until mobile finds there is an empty place, then the acknowledgement will put into  $b3$  immediately without any time delay. We use urgent transition to express this scenario. Since the maximal transmission time is  $TR$ , the maximal waiting time for a free space is  $TR$ .

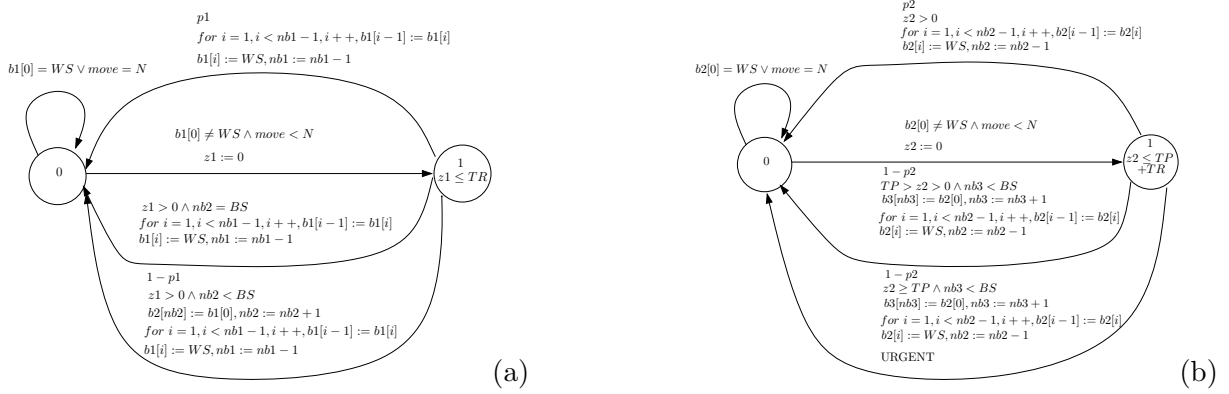


Figure 6: Ch1 and mobile probabilistic timed automata

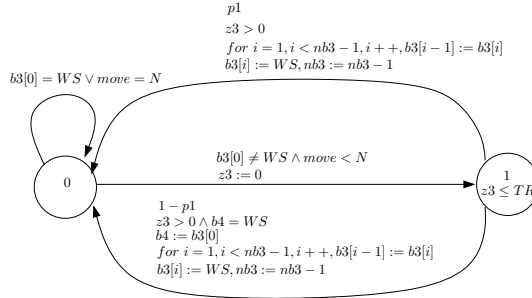


Figure 7: Ch2 probabilistic timed automata

## 4 Prism Model and Verification

### 4.1 Model in Prism

In this section, we give a brief explanation of our model in PRISM and its model checking results by this tool. PRISM is a probabilistic model checker, a tool for the modeling and analysis of systems which exhibit probabilistic behaviors. It is based on the construction of a precise mathematical model specified like reactive modules formalism of Alur and Henzinger [15]. The model is composed of a set of modules which can interact with each other, while each module generally comprises a set of states, representing all the possible configurations of the system, the transitions that can occurs between these states. Properties of the system are then expressed in PCTL or CSL.

Because there are both probabilistic (message lost) and non-deterministic behaviors (transmission time delay and processing time delay) in this system, the modeling formalism we use in PRISM is Markov decision processes (MDP) which allows expression of these behaviors. As before, the model consists of four modules, PC, channel 1, mobile and channel2, which are

communicated by global variables and are synchronized by commands that are labeled with the same action to force two or more modules to make transitions simultaneously.

Each module has the same function as the corresponding automaton we introduced before. Since RRISM does not support arrays, in PRISM model any variable array we use previously in probabilistic timed automata shall change to a set of variables depending on the size of the array. For instance, buffer  $b1$  with buffer size five in our former model can be expressed in PRISM by five variables  $b10, b11, b12, b13,$  and  $b14$ . In the download system, it is common to have several timers recording time passing simultaneously for different purpose, such as time-delay for sending another frames, time passing since sent some frame, time passing for frame transmission and frame processing, etc. Assuming that all the timers elapse at the same clock speed, we can model time elapsing in a synchronous way. Hence it is required that the four modules have commands labeled with the same *time* action to denote time proceeds 1 unit, this causes all the modules to make time progressing synchronously. Below is the *time* action which labels many of the commands in the four modules. For the detailed specification in PRISM, see appendix.

```

module PC
...
[]l0=0& move=0&nb1<=WS->(l0'=1)&(z'=0)&(b10'=3)&(nb1'=nb1+1);
[]l0=0 & move=N->(l0'=10);
[time]l0=1&z<T0&b4=WS->(z'=min(z+1,T0));
[]l0=1&z=T0&b4=WS->(l0'=0); []l0=1 &
b4=3->(l0'=2)&(z'=TD)&(b4'=WS);

//te1 store frame to be sent
[]l0=2 & z=TD & move<N & nb1<=WS & (b4=WS) -> (l0'=3)&(z'=0);
[time]l0=2 & z<TD & b4=WS & use0=0 & use1=0->(z'=min(z+1,TD));//
[time]l0=2&z<TD&b4=WS&use0=1&use1=0&t0<T0->(z'=min(z+1,TD))&(t0'=min(t0+1,T0));//
[time]l0=2&z<TD&b4=WS&use1=1&use0=0&t1<T0->(z'=min(z+1,TD))&(t1'=min(t1+1,T0));
[time]l0=2&z<TD&b4=WS&use0=1&use1=1&t0<T0&t1<T0->(z'=min(z+1,TD))&
(t0'=min(t0+1,T0))&(t1'=min(t1+1,T0)) ;
...

module CH1 z1:[0..TR]; c10:[0..1];
//l0-init, l1-TR,
[time]c10=0 & (b10=WS|move=N)->(c10'=0);
[]c10=0&b10!=WS&move!=N->(c10'=1)&(z1'=0);
//TIME PASSAGE
[time]c10=1 & z1<TR ->(z1'=min(z1+1,TR));
...

module MOBILE
z2:[0..TP+TR];//
clock d10:[0..1];//location
//l0-init, l1-TR,
[time]d10=0& (b20=WS|move=N)->(d10'=0);
[]d10=0&b20!=WS&move!=N->(d10'=1)&(z2'=0);
//TIME PASSAGE
[time]d10=1 & z2<TP->(z2'=min(z2+1,TR+TP));
[time]d10=1 & z2>=TP&nb3=BS&z2<(TP+TR)->(z2'=min(z2+1,TR+TP));
...

module CH2
z3:[0..TR];//clock
e10:[0..1];//location

```

```
//l0-init, l1-TR,
[time]e10=0& (b30=WS|move=N)->(e10'=0);
[!e10=0&b30!=WS&move!=N->(e10'=1)&(z3'=0);
//TIME PASSAGE
[time]e10=1&z3<TR->(z3'=min(z3+1,TR));
...
```

## 4.2 Experimental Results

In this section, we outline the experimental performance results analyzed in PRISM.

**Finish downloading** Our first concern is to let all the frames successfully reach PC, such property is expressed in PRISM as:

$$(1) \quad Pmax = ?[true U(move = N)], Pmin = ?[true U(move = N)]$$

With parameters as: BS=2, WS=2, N=5, TO=12, TD=2, TR=2, TP=2, it shows that both  $Pmax$  and  $Pmin$  equals 1. However, since there are message lost probability, in fact the expected time to satisfy this property is infinite.

**Timeout value effect** One of the purposes investigating this system is to finish downloading the software from PC to mobile as quick as possible. Since the expected time to finish the downloading process with probability 1 is infinite, it is meaningful to consider such property as: the probability to finish downloading the program within expected time  $tt$ . The expected time in this case is the exact time elapsing since sending until finishing it, not the number of discrete time steps. Therefore to calculate  $tt$ , we add another module which is synchronized with other modules via command labelled with `time`.

```
module CalculateT
tt:[0..110];
[time]tt<110->(tt'=tt+1);
[time]tt>100->(tt'=110);
endmodule
```

Property is expressed as below where  $t$  is a global variable recording time since start to send the program.

$$(2) \quad Pmax = ?[true U(t \leq tt) \wedge (move = N)], Pmin = ?[true U(t \leq tt) \wedge (move = N)]$$

Normally, as hardware is fixed which means USB related parameters does not change, we might seek to other parameters to observe their performance effect. The constant  $TO$  plays a crucial

role in the protocol performance, no matter how to change it we need guarantee it to be chosen to allow enough time for a frame to get to the receiver, for the receiver to process it, and for the acknowledgment frame to propagate back to PC in the worst case. Under this condition, figure (8) illustrates the impacts of timeout period on the probability to finish downloading a software within 80 minutes.



Figure 8:  $TO$  changes

The figure shows that if  $TO \leq tt$ , a higher timeout values  $TO$  can decrease the probability of finishing the software downloading within  $tt$ . While when  $TO$  is closer to  $tt$ , it appears that the probability does not change while  $TO$  is increasing. To understand this result, consider the scenario that there is frame or acknowledgment lost. Though there is chances that this frame will be resent immediately without let time elapsing  $TO$  time unit as soon as the acknowledgment of the frames stored in the later position of the window is received, there are cases where PC still need wait  $TO$  to resend this frame. In these cases, increasing  $TO$  need to wait longer time for PC to re-send this frame, which inevitably decreases the probability of ending the download process within time  $tt$ . As  $TO \geq tt$ , the set of pathes satisfying property in formula (2) excludes the set of pathes (including time proceeding transition also) which lead to those states satisfying  $move = N \wedge t \geq tt$ . Therefore, though  $TO$  is on the increase, the probability does not change accordingly.

**Message lost effect** See figure (9), it is easy to understand that as  $p1$  increases,  $P_{max}$  and  $P_{min}$  increases accordingly.

**Window size effect** See figure (10), it shows that window size increase may speed up the download procedure for the best case. However, if window size is large and  $TD$  is not large enough, there will be high rate of frame lost because of  $b2$  buffer overflowing, which will decrease the download speed, and we can see this phenomenon from the change of the minimal probability as  $WS$  increases.

Figure 9:  $p1$  changesFigure 10:  $WS$  changes

**Buffer overflowing** As said in section 2.2, one big problem ever occurred is the software can not be downloaded. The reason lies in that buffer size  $BS$  is so small that most of the frames transmitted to buffer  $b2$  are lost because of the buffer overflowing, we call this *artificial message lost* to distinguish it from physical message lost, and introduce parameter  $TD$  to solve this problem. However, experimental result in PRISM shows that if  $TD$  is not big enough, there is still possibilities that this kind of message lost appears. For instance, with parameters as:  $BS = 1, WS = 2, N = 5, TD = 1, TO = 15, TR = 2, TP = 2$ , model checking property  $Pmax = ?[trueUfull = 1]$  and get the maximal probability of *artificial message lost* in  $b2$  is not zero. However, when increase  $TD$  to two or more, the maximal probability becomes zero. The explanation, in an intuitive way, is that the speed of sending frame is slower than that of processing it which can always cause free space in  $b2$ . Nevertheless, the disadvantage of  $TD$  increase is that this will slow the downloading procedure. Therefore, we need choose  $TD$  as small as possible based on the condition that artificial message lost case does not occur or even occurs the rate is very low. As  $TP$  increases, the experimental results show that the  $b2$  overflowing probability increases accordingly. The reason is that increase of this parameter will let frames staying in  $b2$  longer, which increases buffer overflow probability. Besides these, we also find that the rate increases as  $WS$  increases, while high buffer( $b2$ ) overflowing probability will generate the observation scenario that the software download process does not proceed or proceeds at very slow speed.

There is another way to avoid this kind of message lost scenario, which requires the change of buffer size. Increasing buffer size exceeds or equals the window size  $WSD$ , we found that  $P_{max} = [true \ U_{full} = 1]$  becomes zero. There are two items together to explain this phenomenon. The first item is that the maximal number of frames for buffer  $b2$  can hold is the number of frames in the sending sliding window. Moreover the second point is under the precondition that  $TO$  is big enough to allow for an acknowledgment to propagate back, so that the corresponding frame in the sending window can not have a duplicate one in  $b2$  at any time. These two items combined implies that there could have maximal  $WSD$  frames in  $b2$ , but there are no other frames in  $b1$ , which can not lead to artificial message lost. From this we conclude that artificial message lost in buffer  $b2$  can be avoided when buffer size is equal to or greater than window size.

## 5 Conclusion

Aimed to improve the performance of the PC-mobile download system, in this paper we have given a formal analysis of this system. Since timed and probabilistic information are included in it, we use a network of probabilistic timed automata to give a clear description. The model is built with consideration of state space reduction and timer scheduling. Later, the slightly changed model is analyzed in PRISM, and performance effect with change of system parameters is illustrated.

Though timed information is characterized by digital clock [16], there is still limitation as to the size of the model. Future work includes abstracting the model, for instance using one timer to record all the frame sending information instead of the current method, which is instructive to reduce state space and more easily implementable in industry. Furthermore, based on the abstract model we would like to use the techniques proposed in paper [12] to analyze some specific duration calculus property, such as: for any observation time, eg., 2 time units, the probability of artificial message lost will be less than 0.1.

**Acknowledgment.** We thank Dave Parker from University of Birmingham for very helpful discussions on using the PRISM tool.

## References

- [1] R. Alur and D.L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, pages 183–235, 1994.
- [2] Marta Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.

- [3] Yong Deng and Zhangqin Huang. Modeling and Performance Analysis of a Sliding Window Protocol. Proceedings PROGRESS2003
- [4] Hans Hansson and Bengt Jonsson. *A Logic for Reasoning about Time and Reliability. Formal Aspects Computing*. Vol 6, 512-535, 1994.
- [5] Karsten Stahl, Kai Baukus, Yassine Lakhnech and Martin Steffen. Divide, Abstract, and Model-Check. *Proceedings of the 5th SPIN Workshops on Theoretical Aspects of SPIN Model Checking* Pages: 57 - 76. LNCS 1680, 1999.
- [6] Dmitri Chklyaev, Jozef Hooman and Erik de Vink. Verification and Improvement of the Sliding Window Protocol. *Proc. 9th Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)* ,LNCS 2619, Springer-Verlag, pages 113-127, 2003.
- [7] M. Zhang and F. Vaandrager. Analysis of a protocol for dynamic configuration of IPv4 link local addresses using Uppaal. Technical report, NIII, Radboud University Nijmegen, 2005. To appear.
- [8] M. Kwiatkowska, G. Norman and D. Parker. PRISM 2.0: A Tool for Probabilistic Model Checking. In *Proc. 1st International Conference on Quantitative Evaluation of Systems (QEST'04)*, pages 322-323, IEEE Computer Society Press. September 2004.
- [9] M. Kwiatkowska, G. Norman and D. Parker. Prism 2.1 Users' guide. <http://www.cs.bham.ac.uk/dxp/prism/doc/manual.pdf>
- [10] A. S. Tanenbaum. Computer Networks. Prentice-Hall, 1996.
- [11] M. Hendriks. Model Checking the Time to Reach Agreement. *International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'05)*, to appear.
- [12] Dang Van Hung and Zhang Miaomiao. On Verification of Probabilistic Timed Automata against Probabilistic Duration Properties. Technical Report 326, UNU-IIST, P.O.Box 3058, Macau, June 2005
- [13] M. Dufлот, M. Kwiatkowska, G. Norman and D. Parker. A Formal Analysis of Bluetooth Device Discovery. In *Proc. 1st International Symposium on Leveraging Applications of Formal Methods (ISOLA'04)*
- [14] S. Cheshire, B. Aboba, and E. Guttman. Dynamic configuration of IPv4 link-local addresses, 2004. <http://www.ietf.org/internet-drafts/draft-ietf-zeroconf-ipv4-linklocal-17.txt>.
- [15] Rajeev Alur, Thomas A. Henzinger. Reactive Modules. *Formal Methods in System Design*. 15(1): 7-48 (1999)
- [16] Marta Z. Kwiatkowska, Gethin Norman, David Parker, Jeremy Sproston. Performance Analysis of Probabilistic Timed Automata using Digital Clocks. In *Proc. Formal Modeling and Analysis of Timed Systems (FORMATS'03)*.105-120.

## A Code in PRISM

```

//ws is the variable of WSD in the paper

nondeterministic //MDP

const int BS=2; //buffer size for b2,b3 and b4
const int N=5; //number of frames
const int WS=2;//window size
const int TD=2; //time delay for sending consecutive two frames
const int TO=25; //time out for sending the same frame
const int TR=2; //transmission time delay
const int TP=2; //processing time in mobile
const double p1=0.01;//message lost probability
const double p2=0.05;//bad frames probability
//buffer for b1,b2 and b3ddd
global b10: [0..WS+1] init WS;//b10 buffer size
global b11: [0..WS+1] init WS;
global b12: [0..WS+1] init WS;
global b20: [0..WS+1] init WS;
global b21: [0..WS+1] init WS;
global b22: [0..WS+1] init WS;
global b30: [0..WS+1] init WS;
global b31: [0..WS+1] init WS;
global b32: [0..WS+1] init WS;
global b4:[0..WS+1] init WS;
global nb1: [0..BS] ;//number of places in buffer b1 used.
global nb2: [0..BS]; //number of places in buffer b2 used.
global nb3: [0..BS];//number of places in buffer b3 used.
global ws:[0..WS] init WS;
global move:[0..N];

module PC
z:[0..TO]; //clock for timerdelay
l0:[0..7];//locations
sent0: [0..1]; //boolean to indicate frame in position 0 sent or not.
sent1: [0..1];
ack0: [0..1]; //boolean indicate frame in position 0' ack received or not.
ack1: [0..1];
use0: [0..1]; //boolean indicate clock 0 used or not. the number of clock is WS
use1: [0..1];
real0:[0..1] init 0;//1st frame number in the window,
real1:[0..1] init 1;//2nd frame number in the window

```

```

te1:[0..WS-1] ;//temp variable
t0:[0..TO];//clock 0
t1:[0..TO];//clock 1

[]l0=0& nb1=BS->(l0'=10);
[]l0=0& move=0&nb1<=BS-1->(l0'=1)&(z'=0)&(b10'=3)&(nb1'=nb1+1);
[time]l0=0 & move=N->(l0'=10);
[time] l0=1& z<TO & b4=WS->(z'=min(z+1,TO));
[] l0=1 & z=TO & b4=WS -> (l0'=0);
[]l0=1 & b4=3 -> (l0'=2) & (z'=TD)&(b4'=WS);

//te1 store frame to be sent
[]l0=2 & z=TD & move<N & nb1<=BS-1 &(b4=WS) -> (l0'=3)&(z'=0);
[time]l0=2 & z<TD & b4=WS & use0=0 & use1=0->(z'=min(z+1,TD));
[time]l0=2 & z<TD & b4=WS & use0=1 & use1=0 & t0<TO->(z'=min(z+1,TD))&(t0'=min(t0+1,TO));
[time]l0=2 & z<TD & b4=WS & use1=1 & use0=0& t1<TO->(z'=min(z+1,TD))&(t1'=min(t1+1,TO));
[time]l0=2 & z<TD & b4=WS & use0=1 & use1=1&t0<TO & t1<TO->(z'=min(z+1,TD))
&(t0'=min(t0+1,TO))&(t1'=min(t1+1,TO)) ;
[]l0=2 & z=TD & move<N & nb1=BS&b4=WS -> (z'=0) & (l0'=10);
[]l0=2 & move=N ->(l0'=0);

//sent
[]l0=3 &ws=2& sent0=0 ->(l0'=4) & (sent0'=1)& (te1'=real0);
[]l0=3 &ws=2& sent0=1 & sent1=0 ->(l0'=4) & (sent1'=1)&(te1'=real1);
[]l0=3 &ws=1& sent0=0 ->(l0'=4) & (sent0'=1)& (te1'=real0);

//in case all the frames have been sent
[]l0=3 & sent0=1&sent1=1 &ws=2 ->(l0'=2)&(z'=0);
[]l0=3 & sent0=1 &ws=1 ->(l0'=2)&(z'=0);

[]l0=4 & te1=0->(use0'=1)&(l0'=5)&(t0'=0);
[]l0=4 & te1=1->(use1'=1)&(l0'=5)&(t1'=0);

// which positon in b1 is empty.
[]l0=5 & nb1=0 ->(b10'=te1)&(nb1'=nb1+1)&(z'=0)& (l0'=2);
[]l0=5 & nb1=1 ->(b11'=te1)&(nb1'=nb1+1)&(z'=0)& (l0'=2);

//ws=2 receive ack1=1
[]l0=2& ws=2& b4=0 & real1=0 ->(l0'=6)&(ack1'=1)&(b4'=WS)&(use0'=0)&(t0'=0);
[]l0=2& ws=2 & b4=1 & real1=1->(l0'=6)&(ack1'=1)&(b4'=WS)&(use1'=0)&(t1'=0);
[]l0=6 & ws=2 & ack0=0->(l0'=7)&(sent0'=0);

//ack0=1

```

```

[]l0=2& ws=2& b4=0 & real0=0 ->(l0'=7)&(ack0'=1)&(b4'=WS) &(use0'=0)&(t0'=0);
[]l0=2& ws=2& b4=1 & real0=1->(l0'=7)&(ack0'=1)&(b4'=WS)&(use1'=0)&(t1'=0);
//ws=1 and ws!=1 should deal with separately.
[]l0=2& ws=1 & b4=0 & real0=0 ->(l0'=7)&(ack0'=1)&(b4'=WS)&(use0'=0)&(t0'=0);
[]l0=2& ws=1 & b4=1 & real0=1 ->(l0'=7)&(ack0'=1)&(b4'=WS)&(use1'=0)&(t1'=0);

//ws=1 receive ack0=1
[]l0=7 & ack0=1& move<(N-ws)&move<N ->(l0'=7)&(real1'=real0)&(real0'=real1)
      &(sent0'=sent1)&(ack0'=ack1)&(sent1'=0)&(ack1'=0)&(move'=move+1);
[]l0=7 & ws=2& ack0=1& move=(N-ws)&move<N ->(l0'=7)&(real0'=real1)&(sent0'=sent1)
      &(ack0'=ack1)&(move'=move+1)&(ws'=ws-1);
[]l0=7 & ws=1& ack0=1& move=(N-ws)&move<N ->(l0'=7)&(move'=move+1)&(ws'=ws-1);
[]l0=7 & (ack0=0 | move=N)->(l0'=2);

//ws=2,timeout need reset clock 0
[]l0=2 &ws=2 & t0=T0 & real0=0->(sent0'=0)&(l0'=2)&(use0'=0)&(t0'=0);
[]l0=2 &ws=2 & t0=T0 & real1=0->(sent1'=0)&(l0'=2)&(use0'=0)&(t0'=0);
//timeout need reset clock 1
[]l0=2 &ws=2 & t1=T0 & real0=1->(sent0'=0)&(l0'=2)&(use1'=0)&(t1'=0);
[]l0=2 &ws=2 & t1=T0 & real1=1->(sent1'=0)&(l0'=2)&(use1'=0)&(t1'=0);
//ws=1
[]l0=2 &ws=1 & t0=T0 & real0=0->(sent0'=0)&(l0'=2)&(use0'=0)&(t0'=0);
[]l0=2 &ws=1 & t1=T0 & real0=1->(sent0'=0)&(l0'=2)&(use1'=0)&(t1'=0);
endmodule

module CH1
z1:[0..TR];
c10:[0..1];
full:[0..1];
//l0-init, l1-TR,
[time]c10=0 & (b10=WS|move=N)->(c10'=0);
[]c10=0&b10!=WS&move!=N->(c10'=1)&(z1'=0);
//TIME PASSAGE
[time]c10=1 & z1<TR ->(z1'=min(z1+1,TR));
//send from b1 to b2
[]c10=1&z1>0&nb2=0&nb1>=1 ->(1-p1):(c10'=0)&(b20'=b10)&(nb2'=nb2+1)&
      (b10'=b11)&(b11'=WS)&(nb1'=nb1-1)
      + p1:(c10'=0)&(b10'=b11)&(b11'=WS)&(nb1'=nb1-1);
[]c10=1&z1>0& nb2=1&nb1>=1->(1-p1):(c10'=0)&(b21'=b10)&(nb2'=nb2+1)&
      (b10'=b11)&(b11'=WS)&(nb1'=nb1-1)
      +p1:(c10'=0)& (b10'=b11)&(b11'=WS)&(nb1'=nb1-1);
[]c10=1& z1>0 & nb2=2&nb1>=1->(c10'=0)&(b10'=b11)&(b11'=WS)&(nb1'=nb1-1)&(full'=1);
      //b2 full, message lost
endmodule

```

```

module MOBILE
z2:[0..TP+TR];//clock
d10:[0..1];//location
//l0-init, l1-TR,
[time]d10=0& (b20=WS|move=N)->(d10'=0);
[]d10=0& b20!=WS& move!=N->(d10'=1)&(z2'=0);
//TIME PASSAGE
[time]d10=1 & z2<TP->(z2'=min(z2+1,TR+TP));
[time]d10=1 & z2>=TP&nb3=BS&z2<(TP+TR)->(z2'=min(z2+1,TR+TP));
//from b2 to b3
[]d10=1&z2>0&z2<TP & nb3=0 &nb2>=1->(1-p2):(d10'=0)&(b30'=b20)&(nb3'=nb3+1)&
    (b20'=b21)&(b21'=WS)&(nb2'=nb2-1)
    + p2:(d10'=0)&(b20'=b21)&(b21'=WS)&(nb2'=nb2-1);
[]d10=1& z2>0&z2<TP& nb3=1&nb2>=1->(1-p2):(d10'=0)&(b31'=b20)&(nb3'=nb3+1)&
    (b20'=b21)&(b21'=WS)&(nb2'=nb2-1)
    +p2: (d10'=0)&(b20'=b21)&(b21'=WS)&(nb2'=nb2-1);
[]d10=1&z2>=TP & nb3=0 &nb2>=1->(1-p2):(d10'=0)&(b30'=b20)&(nb3'=nb3+1)&
    (b20'=b21)&(b21'=WS)&(nb2'=nb2-1)
    + p2:(d10'=0)&(b20'=b21)&(b21'=WS)&(nb2'=nb2-1);
[]d10=1& z2>=TP& nb3=1&nb2>=1->(1-p2):(d10'=0)&(b31'=b20)&(nb3'=nb3+1)&
    (b20'=b21)&(b21'=WS)&(nb2'=nb2-1)
    +p2: (d10'=0)&(b20'=b21)&(b21'=WS)&(nb2'=nb2-1);
endmodule

module CH2
z3:[0..TR];//clock
e10:[0..1];//location
//l0-init, l1-TR,
[time]e10=0& (b30=WS|move=N) ->(e10'=0);
[]e10=0&b30!=WS&move!=N->(e10'=1)&(z3'=0);
//TIME PASSAGE
[time]e10=1&z3<TR->(z3'=min(z3+1,TR));
//send from b3 to b4
[]e10=1&z3>0 & b4=WS & nb3>=1->(1-p1): (e10'=0)&(b4'=b30)&(b30'=b31)&(b31'=WS)
    &(nb3'=nb3-1)+p1:(e10'=0)&(b30'=b31)&(b31'=b32)&(b32'=WS)&(nb3'=nb3-1);
endmodule

```