Formal Analysis of Streaming Downloading Protocol for System Upgrading

Miaomiao Zhang *, 1

School of Software Engineering
Tongji University
Shanghai, China

Dang Van Hung 2

International Institute of Software Technology
United Nations University
Macau, China

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 message loss and unordered data using a probabilistic model checking tool PRISM. We introduce a technique to reduce the state space of the system modeling 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.

Key words: Streaming download protocol, Probabilistic model checking, Prism.

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 that software could not be downloaded or that it takes unacceptably long time to

1 Email: miaomiao@mail.tongji.edu.cn
2 Email: dvh@iist.unu.edu

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

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 (universal serial bus) connection.

The mechanism of this protocol is somewhat 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 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 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 is a lot of space for the memory in the mobile compared with that in the USB buffer. Moreover in this paper we focus on the download performance analysis while we pay less attention to the processing of data frames on the mobile side.

An important expected property of this system is time related: that the software downloading can be finished as quickly as possible. To measure the downloading time, we introduce timing information parameters for this system model. The typical ones are frames sent interval and timeout values on the PC side, transmission time delay for the USB, and processing time on the mobile side. The protocol exhibits timing and nondeterministic behaviors. Besides, one key point difference between the system in this paper and the ones in other sliding window papers [5,6,10] is that in this system there is likelihood of some transitions, for instance transitions with respect to frame loss, and unordered frame probability are taken into account. A natural model that embodies the nondeterministic, 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 on the mobile side can take values nondeterministically but are bounded by some constant, and (ii) each frame sent that has not been received needs a timer to record the time elapsed since it was sent, as a base to decide whether or not to resend the frame, (iii) all the clocks used in the system evolve 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 [15,7,17] 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 itself from others, while for modelling this could lead to state explosion. To avoid a 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 the sliding window. This suffices to guarantee the unique identification of each frame in this window. When the window moves, some frames at the left of the window end up outside of the window, and the same number of new frames enter from the right of the window. The labels of the frames which leave the window from the left are then reused for the new frames entering the window. A window movement accumulation variable *move* indicates the number of frames having been downloaded successfully. So, combined with *move*, 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 is associated with a timer, the timer assignment and release also need to be carefully considered 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 [3]. However, the analysis in our paper is more formal. Moreover, the algorithm, the behaviors and the mechanism presented in our system are more complete and closer to reality. Our main contribution of this paper is to give a formal model for the downloading system described above 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 details of PRISM see [8] and [9]) regarding the change of values of different parameters show the qualitative property, which can give us a clear explanation of the problems encountered in industry, and are helpful to identify optimal parameters settings and to determine a number of key parameters to improve system performance such as the timeout period, transmission time delay and processing time.

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 encountered in practice for the protocol. In Section 3, the concept of probabilistic timed automata is recalled, followed by the analysis and construction of the abstract system model for the download protocol as 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 Section 5 is the conclusion of the paper.

2 PC-Mobile System and Download Protocol

In this section, we give an overview of the download protocol and list some problems arising within the industry field.

2.1 Overview of the system

Information transferred by this protocol is organized into packets contained within frames. A frame contains one packet which has been encoded for transmission over a particular link. A link 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 which is a type of plug-in connection that is used to connect devices for PC and mobile (see Fig. 1, where channel 1 and channel 2 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, $b1$, $b2$, $b3$ and $b4$ where $b1$ is PC USB sending buffer, $b2$ is mobile USB receiving buffer, $b3$ is mobile USB sending buffer and $b4$ is PC USB receiving buffer. Buffers $b1$ and $b2$ store frames, $b3$ and $b4$ store acknowledgments. At a time, a frame is first sent to $b1$, then channel 1 transmits it from $b1$ to $b2$. After processing the frame on the mobile side, the mobile may generate a corresponding acknowledgment to $b3$, then channel 2 transmits the acknowledgment to $b4$, and the PC can issue other actions according to the mobile response. USB is 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 lead to frame loss and unordered frames. We use $p1$ to denote the probability of message loss in the channels, and $p2$ to denote the probability of frame reordering found on the mobile side.

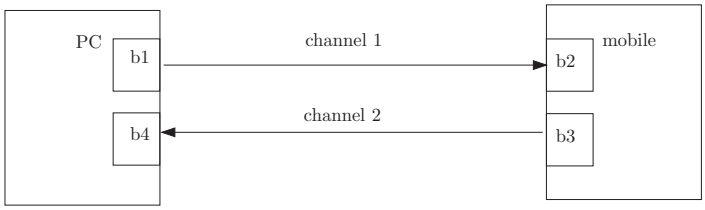


Fig. 1. Downloading system

To download software from the PC, first the PC sends a *hello* frame through wire channel 1 to the mobile to inform the mobile that it will start to transfer software, and at the same time, the PC starts a timer z. If the mobile is ready for accepting the data, it will send an $AckH$ back to the PC through wire channel 2. Otherwise, the mobile sends nothing. In case the mobile is not ready for receiving data, or the message *hello* or $AckH$ is lost, the timer z will generate a timeout, which means that the PC has been waiting a certain length of time for the acknowledgment after sending a frame without receiving

it. The PC then needs to retransmit the *hello* frame again. As soon as 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 flash memory in the mobile to which the frame will be written, in this way different frames can be distinguished by different target address of the flash memory 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 discards 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 acknowledgment $ack[i]$ which is sent back to the PC including the relative address (label) of the frame i. In 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. 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. The PC will not wait for an acknowledgment frame before sending another frame in the window, provided that the cursor have not reached the right end of the window. Note here that if all the frames in the window are sent, the PC will wait until the acknowledgment for the leftmost frame is received. If the PC receives an acknowledgment, it may assume that any unacknowledged frames sent before the one being acknowledged were either lost or failed and needs to send them again. This implies that if the acknowledgment does not correspond to the leftmost one, the window will not move. If the acknowledgment corresponds to the leftmost one, the window will move right by one frame. This process continues until the entire program has been transferred.

For the mobile side, after it gets a frame from $b2$, it will process the frame by checking if it is good or not, and then generate an acknowledgment which will be sent back to the PC if the frame is correct. The successfully received frames from the PC side are temporarily stored in the RAM of the mobile, and then put into the flash memory after every fixed time interval, according to the address within the frames.

2.2 Problems found

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

- The download process does not proceed, the software is not downloaded at all.
- The software is downloaded, but at a very low speed. The question is how

to change protocol parameters to speed up the downloading.

After carefully checking, people in that company found that due to the very limited buffer size in $b2$ and the fast sending of frames from the PC, in fact most of the frames were lost because of the slow data processing of the mobile that lead to the overflow of $b2$ buffer. To solve the problem, in the protocol implementation they added another parameter TD being a time delay between the sending of two consecutive frames and hereby solved the first problem. However, the TD they set is big and it took several minutes to download a small program. Although, they have not yet proved the solution in a formal way and it is still not clear if the TD they set is rational. Moreover, there are a number of parameters in the protocol, and questions raised as to how the parameters affect the performance and how to make the downloading as fast as possible need to be further investigated.

One should note here that for the PC-Mobile system implemented in the company, compared with the USB buffer there is enough RAM memory on the mobile side, and only after checking that there is room in buffer $b3$ can the acknowledgment arrive at it, therefore there is no buffer overflow 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 are listed as follows.

- TD, as said before, is the time delay between sending two consecutive frames.
- TO is the time out value, from the the time moment the PC sends a frame, until the time moment to resend the frame if no acknowledgment for this frame has been received.
- TR is the maximal transmission time for a frame or an acknowledgment through channel 1 or channel 2.
- TP is the maximal processing time of a frame by the mobile. The purpose for introducing this parameter is also to analyze the two problems mentioned above. If the processing of a frame on the mobile side is much faster than sending a frame to channel 1 on the PC side, obviously buffer $b2$ overflow 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 $b3$ (not $b4$) 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 the software to be downloaded) and the size WS of the sliding window.

The complete download system consists of not only timing constraints, but also stochastic information. For modeling such kind of systems, we use probabilistic timed automata, which we considered to be most suitable for our purpose.

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 well as its semantics from [2].

Probability distributions and Markov decision processes

A probability distribution over a set S is a mapping $p : S \to [0, 1]$ such that the set $\{s|s \in S \text{ and } p(s) > 0\}$ 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 $Steps : \mathcal{Q} \to 2^{\mu(\mathcal{Q})}$ 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 Σ, and assume that $Steps : \mathcal{Q} \to 2^{\Sigma\times\mu(\mathcal{Q})}$. So, a transition is of the form $q \xrightarrow{a,p} q'$, where $(a, p) \in \Sigma \times \mu(Q)$ is the label of the transition. We also assume a la belling function $L : \mathcal{Q} \to 2^{AP}$, 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 la belled Markov decision process is a tuple $(\mathcal{Q}, Steps, L)$.

La belled paths (or execution sequences) are nonempty finite or infinite sequence of consecutive transitions on 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 ω, let $first(\omega)$ denote the first state of ω, and if ω is finite then let $last(\omega)$ denote the last state of ω. $|\omega|$ is the length of ω and is defined as the number of transition occurrences in ω which is ∞ if ω is infinite. For $k \le |\omega|$, let $\omega(k)$ denote the kth state of ω, and $step(\omega, k)$ denote the label of the kth transition in ω. 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 ω and ω' 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 $\nu : \mathcal{C} \to \mathbb{R}$ 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 ν 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 ν 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 ζ and a set of clocks $X \subseteq \mathcal{C}$ the set $\{\nu[X := 0] | \nu \in \zeta\}$ 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 3.1 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 $\mathcal{L} : \mathcal{S} \to 2^{AP}$ 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 $inv : \mathcal{S} \to \mathbf{Z}_{\mathcal{C}}$ assigning to each node an invariant condition,
- a function $prob : \mathcal{S} \to 2^{\mu(\mathcal{S} \times 2^{\mathcal{C}})}$ 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 $s \in \mathcal{S}$, $\tau_s : prob(s) \to \mathbf{Z}_{\mathcal{C}}$ 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 in 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 $\mathbf{Z}_{\mathcal{C}}(G)$ the set of all clock zones occurring in G,

$$\mathbf{Z}_{\mathcal{C}}(G) \quad \{inv(s) \in \mathbf{Z}_{\mathcal{C}} | s \in \mathcal{S}\} \cup \{\tau_s(p) \in \mathbf{Z}_{\mathcal{C}} | s \in \mathcal{S} \text{ and } p \in prob(s)\}.$$

3.2 Modeling Ideas

In this section, we present our ideas and methods for modeling the system, mainly concerning big state space reduction and the processing of related window movement. Firstly we present some assumptions of the system for our later analysis with the PRISM tool.

(i) Message (frame or acknowledgment) sent from one side to the other side arrives in the same order as the one in which they were sent.

(ii) There is a nonzero time delay for the message transmission, i.e. a message sent from one side cannot reach the other side until at least some time has elapsed.

(iii) Since mobile needs to check whether a frame is bad or not, we assume it takes nonzero time for message processing.

(iv) 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.

State space reduction and window movement

In this protocol the number of frames of a software download could be over ten thousands which are distinguished by different addresses and can be la belled by integers. Moreover, for each frame, we need several various auxiliary variables to store related information for this frame including (1) if this frame is sent or not, (2) if sent, which clock records the time passage for its acknowledgment, (3) if the acknowledgment of this frame has been received or not and (4) at which position in the sliding window the frame is located. Therefore even for a small-size software package, the state space could be very big because of the various frames and their related variables. To avoid the state space exposition in model checking, we use variables to differentiate only the frames within the sliding sending window rather than each frame of the software. Suppose the sliding window can hold WS frames and position in the window is expressed by variable i in the range 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 used to indicate the frame at the position i of the window, i.e. whether the frame with the label $real[i]$ has been sent or not. Similarly, the variable $ack[i]$ indicates if the acknowledgment of the frame with the label $real[i]$ has been received or not by the PC.

As soon as the PC gets 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 $ack[j]$ to 1. Window does not move, hence variables regarding window movement do not change.
- If $j = 0$ and starting from position 1 continuously there exist positions whose correspondent acknowledgment have been received. In other words, there exists m, $0 \leq m < WS - 1$, such that $ack[0] = 1 \wedge ack[1] = 1 \ldots \wedge ack[m] = 1$. In this case window moves $m + 1$ steps to the right and the position of the remaining frames in the window is decreased by $m + 1$, while the frame labels in the vector $real$ starting from position 0 until position m are moved to the place with the starting position as $WS - (m + 1)$ and the end position as $WS - 1$ to be reused for the frames that just entered the window. The related variables for the newly entered frames $ack[i]$ and $sent[i]$, $i \in \{WS - (m + 1), \ldots, WS - 1\}$ are set to 0 meaning that these new frames need to be sent. Figure (2) gives the description for this case with $WS = 6$ and $m = 2$ implying that window moves 3 steps.

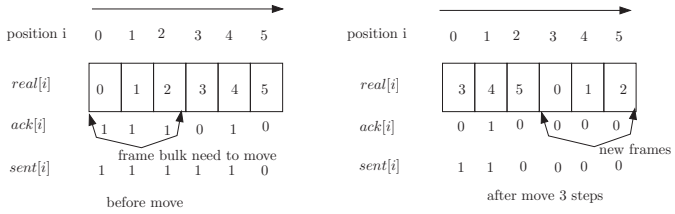


Fig. 2. Window moves 3 steps

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 successfully arrived at the mobile and the 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 remaining frames that have not been transferred is equal to the size of the current window, when moving the current window by one step we will decrease the window size to $WSD - 1$ accordingly.

Timer evaluation

When 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 with the time out TO. When the acknowledgment is received or timer times 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 handling these timers.

We interpret the timer $t[real[i]]$ related to the frame $real[i]$ in the window as a clock in our model where i is the window position. At any time, $t[real[i]]$ indicates the current time of the $real[i]$-th clock. After acknowledgment for

frame $real[i]$ is received within TO time units, or when $t[real[i]]$ expires after TO time units causing the frame $real[i]$ to be resent, we need to reset $t[real[i]]$ to 0. Figure (3) illustrates how $t[real[i]]$ changes after moving the window 3 steps, in this case it should be that acknowledgment for the frame in position 0 is just received.

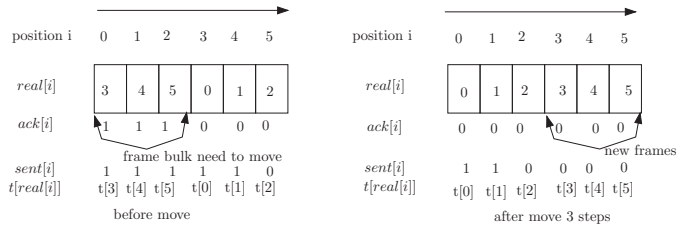


Fig. 3. Timer before and after window moves 3 steps

If the PC receives the acknowledgment for a frame, it assumes that any unacknowledged frame sent before this one was either lost or failed and needs to be sent again, and its relevant clock is reset to 0.

Resending 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 amount of time that elapses. When the timer expires after TO time units without the acknowledgment for this frame received, then the frame needs 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 the frame 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, we can use the pair $(move, real[j])$ as the unique label for the frame.

We describe the procedure for the PC for the protocol formally 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 parallel */
    /*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;
    /*recevied_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)
```

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 frame i	$0...TO$
WSD	current window size equals WS initially	$0...WS$
$move$	steps window has moved	$0...N$
$b1[i]$,$b2[i]$,$b3[i]$	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

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

3.3 Protocol Model

In the following, we describe the PC-mobile download system as a network 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 frames being reordered. As the window size may change when the download procedure nearly completes, 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 whether the frame in position i of the sending window is sent or not. This includes the situation both for sending the frame for the first time and resending the frame no matter how many times it has been sent. The value of any element of buffer arrays $b1$, $b2$, $b3$ and variable $b4$ ranges between 0 and $WS+1$. The values from 0 to $WS-1$ indicate frames or acknowledgments label. While the value WS indicates that there is no frame or acknowledgment stored in this place and it is therefore empty, the value

$WS + 1$ indicates that the frame or acknowledgment stored is related to the *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 the PC is shown in Figure 4. The automaton commences in the location 0. The self-loop transition at the location 0 expresses that the program is successfully downloaded. If the software has not been started to transfer, the PC moves to the location 1 by sending a *hello* frame to buffer $b1$ meanwhile setting up a timer z. As soon as the PC receives the *hello* acknowledgment, it moves to the location 2 to download the software. The *hello* frame needs to be resent when the timer z times out and there is no acknowledgment arrived.

The sequence of transitions from location 2, passing locations 3, 4, 5 and ending in location 2, implements the procedure to send a frame whose position i in the sliding window is the smallest and is indicated as having not been sent, to launch a timer to record time passed since the frame is sent and to find the right place in $b1$ to store this frame.

As the PC finds the acknowledgment i in $b4$, it will immediately handle it. Using the transitions from location 2 to location 6 and to location 7 respectively, we differentiate whether or not the related frame for acknowledgment i is located in the first position of the sliding window. In addition to these, PC needs to resend all these unacknowledged frames sent before the one being acknowledged, which is illustrated by the transition starting from location 6.

The two self-loop transitions centered around location 7 model window movement scenarios. One of them with the guard $move = N - WSD$ implies that window size is decreased by 1 after the movement. If $move$ equals the number of frames N or the acknowledgment for the frame in position 0 is not received, from location 7 the PC transits to location 2.

In the model we also take into account the case when the time delay between sending two consecutive frames is set to 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 the model assumptions this situation does not arise.

The model for channel 1 is shown in Figure 5-a. The automaton starts in location 1, where it resides until there is a frame in $b1$ and moves to location 1. The upper bound for the transmission time delay in channel 1 is TR, according to the model assumptions only when clock $z1$ used to record time elapsed for frame transmission is greater than 0 can outgoing transitions from location 1 take place. When $b2$ is full, the frame coming from $b1$ is lost due to $b2$ buffer overflowing, otherwise frame successfully reaches $b2$ or still gets lost because of physical message loss.

The model for mobile and channel 2 are shown in Figure 5-b and Figure 6 respectively. Note that the invariant in the mobile automaton is $z2 \leq TP + TR$. This is because even though the maximal frame processing time is TP, in the case when $b3$ is full, an acknowledgment will still stay in RAM of mobile

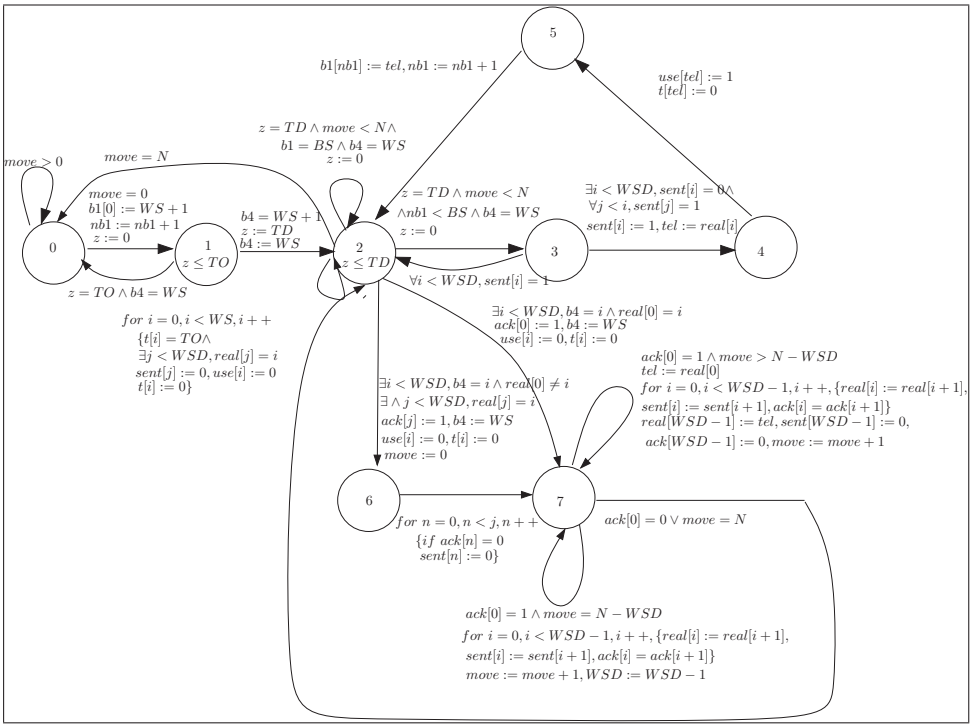


Fig. 4. PC probabilistic timed automaton

and cannot reach $b3$ until mobile finds there is an empty place for it and puts it into $b3$ immediately without time delay. We use an urgent transition to express this scenario. Since the maximal transmission time is TR, the maximal waiting time for a free space is also TR.

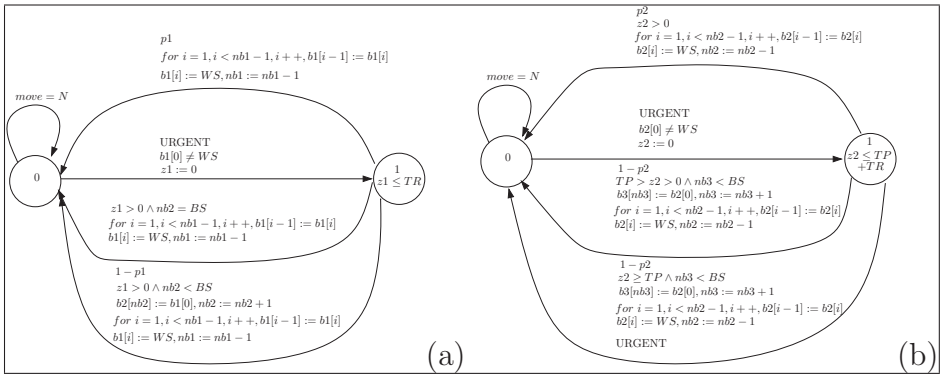


Fig. 5. Channel 1 and mobile probabilistic timed automata

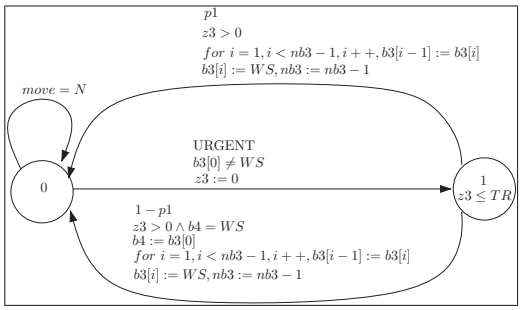


Fig. 6. Channel 2 probabilistic timed automaton

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 [16]. The model is composed of a set of modules which can interact with each other, where each module generally comprises a set of states, representing all the possible configurations of the system and the transitions that can occur between these states. Properties of the system are then expressed in PCTL or CSL.

Because there are both probabilistic (message lost, frame reordered) 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 channel 2, which are communicated by global shared variables and are synchronized by commands that are labelled 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 PRISM does not support arrays, in a 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 purposes, such as time-delay for sending another frame, time passing since some frame was sent, time passing for frame transmission and frame processing, etc. Assuming that all the timers are running at the same rate, we can model time elapsing in a synchronous way. Therefore it is required that the four modules have commands labelled with the same *time* action to denote time proceeds 1 unit, this causes all the modules to make progress

synchronously. Below is the *time* action which labels many of the commands in the four modules. We refer to [12] for the detailed source code in PRISM.

```
module PC
 ...
[]l0=0& move=0&nb1<=WS->(l0'=1)&(z'=0)&(b10'=3)&(nb1'=nb1+1);
[]l0=0 & move=N->(l0'=l0);
[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);

[]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<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));
...

module CH1
z1:[0..TR];//clock
cl0:[0..1]; //location
[time]cl0=0 & (b10=WS|move=N)->(cl0'=0);
[]cl0=0&b10!=WS&move!=N->(cl0'=1)&(z1'=0);
[time]cl0=1 & z1<TR ->(z1'=min(z1+1,TR));
 ...

module MOBILE
z2:[0..TP+TR];// clock
dl0:[0..1];//location
[time]dl0=0& (b20=WS|move=N)->(dl0'=0);
[]dl0=0&b20!=WS&move!=N->(dl0'=1)&(z2'=0);
[time]dl0=1 &z2<TP->(z2'=min(z2+1,TR+TP));
[time]dl0=1 &z2>=TP&nb3=BS&z2<(TP+TR)->(z2'=min(z2+1,TR+TP));
 ...

module CH2
z3:[0..TR];//clock
el0:[0..1];//location
[time]el0=0&(b30=WS|move=N)->(el0'=0);
[]el0=0&b30!=WS&move!=N->(el0'=1)&(z3'=0);
[time]el0=1&z3<TR->(z3'=min(z3+1,TR));
 ...
```

4.2 Experimental Results

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

Finishing downloading

Our first concern is to have all the frames successfully reach PC, this requires $move = N$ and property is expressed in PRISM as:

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

With a set of different parameters, model checking results show that both $Pmax$ and $Pmin$ equal 1. However, since there are message loss probability, in fact in the worse case the time to satisfy this property is infinite.

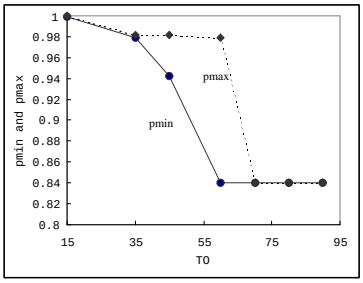
Timeout value effect

One of the purposes of investigating this system is to finish downloading the software from PC to mobile as quickly as possible. Since the expected time to finish the downloading process with probability 1 is infinite, it is meaningful to consider such a property as: the probability to finish downloading the program within expected time tt. As illustrated before, the program is completely downloaded when $move = N$. This property is expressed as below where t is a variable recording time elapsed since the PC started sending the program.

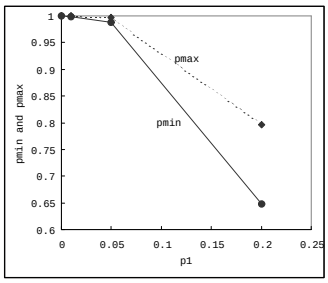
(2) $Pmax = ?[true\ U\ (t \leq tt) \wedge (move = N))],$
$Pmin = ?[true\ U\ (t \leq tt) \wedge (move = N))]$

Variable t in the formula 2 in this case is the exact time passed from sending until finishing, not the number of discrete time steps. Therefore to calculate t, we add another module which is synchronized with four other modules via a command labelled `time`.

Normally, as hardware is fixed which means USB related parameters do not change, we might seek to investigate 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 to guarantee it to be chosen so that it allows enough time for a frame to get to the mobile, for the mobile to process it, and for the acknowledgment frame to propagate back to PC in the worst case. Under this condition, Figure 7-a illustrates the impacts of the timeout periods on the probability to finish downloading a software within 80 time units.



(a)



(b)

Fig. 7. (a) TO changes (b) $p1$ changes

The figure shows that if $TO \leq tt$, with a higher timeout value for TO

we can decrease the probability of finishing the software downloading within tt. Whereas 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 a frame or an acknowledgment lost. Though there exists one case that this frame will be resent immediately without letting time elapse for TO time units, as soon as the acknowledgment of the frames stored in the later position of the window is received, there are cases where the PC still needs to wait TO time units to resend this frame. In these cases, increasing TO to let the PC wait a longer time to resend this frame, inevitably decreases the probability of completing the download process within time tt. As $TO \geq tt$, because the set of paths satisfying property in formula 2 excludes the set of paths (including time proceeding transitions also) which lead to those states satisfying $move = N \wedge t > tt$, therefore, though TO is in the increase direction, the probability does not change accordingly.

Message loss effect

See Figure 7-b, it is easy to understand that as $p1$ increases, $Pmax$ and $Pmin$ decrease accordingly.

Window size effect

Figure 8 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 loss because of buffer $b2$ overflowing, which will decrease the download speed, and we can see this phenomenon from the change of the minimal probability as WS increases.

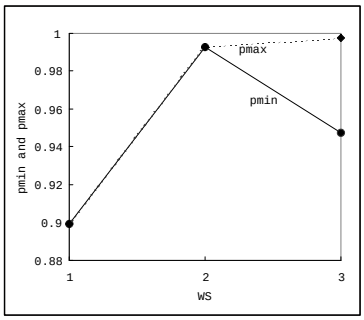


Fig. 8. WS changes

Buffer overflowing

As shown in Section 2.2, one big problem that could occur is that the software could not be downloaded. The reason lies in that the buffer size BS is so small that most of the frames transmitted to buffer $b2$ are lost because of the buffer overflow. We call this *artificial message loss* to distinguish it from physical message loss, and introduce parameter TD to solve this problem. However,

experimental results in PRISM show that if TD is not big enough, there is still a possibility that this kind of message loss appears. For instance, with the parameters setting as: $BS = 1, WS = 2, N = 5, TD = 1, TO = 15, TR = 2, TP = 2$, and with the model checking property $Pmax =?[trueUfull = 1]$, we get that the maximal probability of *artificial message loss* in $b2$ is not zero. When increasing TD to two or more, the maximal probability becomes zero. The explanation, in an intuitive way, is that the speed of sending a frame on the PC side is slower than that of processing it on the mobile side which can always cause free space in $b2$. Nevertheless, the disadvantage of TD increase is that this will slow down the downloading procedure. Therefore, we need choose TD as small as possible based on the condition that *artificial message loss* does not occur or if it occurs the rate is very low. As TP increases, the experimental results show that the $b2$ overflow probability increases accordingly. The reason is that increase of this parameter will let frames stay in $b2$ for longer, which increases buffer overflow probability. Besides these, we also find that the rate increases as WS increases, while high buffer($b2$) overflow probability will generate the scenario that the software download process does not proceed or proceeds at very slow speed.

In addition to introducing TD, there is another way to avoid this kind of message loss scenario, which requires the change of the buffer size. By increasing the buffer size to exceed or equal the initial window size WS, we found that $Pmax =?[true\ Ufull = 1]$ becomes zero. There are two reasons for this phenomenon. The first reason is that the maximal number of frames buffer $b2$ can hold is the number of frames in the sending sliding window. The second reason is, under the precondition that TO is big enough to allow for an acknowledgment to propagate back, that the corresponding frame in the sending window can not have a duplicate one in $b2$ at any time. These two reasons combined imply that there can be a maximum of WS frames in $b2$, but that there are no other frames in $b1$, which can not lead to *artificial message loss*. From this we conclude that the *artificial message loss* scenario in buffer $b2$ can be avoided when the buffer size is equal to or greater than the initial window size.

5 Conclusion

Aiming at improving 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 used a network of probabilistic timed automata to model the system, and gave a clear description. The model is built with consideration of state space reduction and timer scheduling. Later, the model was slightly modified and analyzed in PRISM. The performance effects with the change of system parameters were obtained and are useful for the improvement of the system design.

Though timed information is characterized by digital clocks [17], there is

still a 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 would be instructive to reducing state space and more easily implementable in industry. Furthermore, based on the abstract model we would like to use the techniques proposed in [13] to analyze some specific Probabilistic Duration Calculus properties.

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 PROGRESS 2003.

[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 Chkliaev, 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] B. Gebremichael, F.W. Vaandrager, and Miaomiao Zhang. Analysis of a protocol for dynamic configuration of IPv4 link local addresses using Uppaal. Technical report, NIII, Radboud University Nijmegen, 2006. To appear. Available at *http://www.cs.ru.nl/ita/publications/papers/fvaan/zeroconf/*

[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)*. 98-111.

[12] Zhang Miaomiao and Dang Van Hung. Formal Analysis of Streaming Downloading Protocol for System Upgrading. Technical Report 332, UNU-IIST, P.O.Box 3058, Macau, Dec 2005.

[13] 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.

[14] M. Duflot, 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)*.

[15] 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.

[16] Rajeev Alur, Thomas A. Henzinger. Reactive Modules. *Formal Methods in System Design*. 15(1): 7-48 (1999).

[17] 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.