PAPER *Special Section on Formal Approach*

QoS Analysis of Real-Time Distributed Systems Based on Hybrid Analysis of Probabilistic Model Checking Technique and Simulation

Takeshi NAGAOKA[†a)], **Akihiko ITO**[†], *Nonmembers*, **Kozo OKANO**[†], *and* **Shinji KUSUMOTO**[†], *Members*

SUMMARY For the Internet, system developers often have to estimate the QoS by simulation techniques or mathematical analysis. Probabilistic model checking can evaluate performance, dependability and stability of information processing systems with random behaviors. We apply a hybrid analysis approach onto real-time distributed systems. In the hybrid analysis approach, we perform stepwise analysis using probabilistic models of target systems in different abstract levels. First, we create a probabilistic model with detailed behavior of the system (called detailed model), and apply simulation on the detailed model. Next, based on the simulation results, we create a probabilistic model in an abstract level (called simplified model). Then, we verify qualitative properties using the probabilistic model checking techniques. This prevents from state-explosion. We evaluate the validity of our approach by comparing to simulation results of NS-2 using a case study of a video data streaming system. The experiments show that the result of the proposed approach is very close to that of NS-2 simulation. The result encourages the approach is useful for the performance analysis on various domain.

key words: *QoS, probabilistic automaton, simulation, model checking*

1. Introduction

Nowadays real-time distributed systems like streaming media systems are widely spreading. These systems require time based transmission such as QoS control to prevent interruption of packet transmission caused by network delay, packet loss, and so on. Since the Internet is a best-effort network shared by a number of nodes, there is no guarantee on the QoS properties such as network bandwidth, delay and throughput. Therefore, system developers preliminary have to estimate the QoS by simulation techniques [1] or mathematical analysis [2].

Simulation techniques usually do not guarantee qualitative properties such as the maximum throughput and the minimum jitter, and so on, though they can calculate mean-values along typical traces. In general, these techniques use much resources to simulate accurately the target network systems. On the other hand, mathematical analysis is logically correct, but in many cases the based models are too ideal; hence it is sometimes hard to apply the mathematical analysis to realistic applications.

Formal verification techniques, especially model checking techniques [3] are considered as promising techniques for information systems developing due to their ability of exhaustive checking. Among them, probabilistic model checking can evaluate performance, dependability and stability of information processing systems with random behaviors [4]. PRISM [5] is one of the probabilistic model checkers. It handles automata with probabilities (discrete and continuous time Markov chains) and time elapse. Therefore, it is suitable for modeling the network systems. One of the approaches, which overcomes drawbacks of simulation approach and model checking approach, seems to be a kind of a hybrid approach.

In order to find if the hybrid approach is applicable to real systems, in this paper, we apply a hybrid analysis technique onto real-time distributed systems, which uses both of simulation and model checking techniques. In our approach, we perform a stepwise analysis using probabilistic models of target systems in different abstract levels. First, we create a probabilistic model with detailed behavior of the system (called detailed model), and apply simulation on the detailed model. Next, based on the simulation results, we create a probabilistic model in an abstract level (called simplified model). Then, we verify qualitative properties using the probabilistic model checking techniques.

As the target real-time distributed system, we use an experimental system shown in Fig. 1. The main subsystem of the system is video data streaming which uses RTSP (Real Time Streaming Protocol) for the streaming protocol. The system has also ftp servers and clients which exploit tcp connections, as well as a packet generator that generates udp packets as background noise. Thus, the system involves several simultaneous sessions; it may cause congestion.

In Papers [6]–[8], we have given case studies where we evaluated QoS properties of real-time distributed systems using both of simulation and model checking functions in PRISM. Based on the case studies, this paper summarizes the QoS evaluation technique for real-time distributed systems in general. Especially, we increase the number of scenarios for the example in order to gain degree of accuracy. The results show that our approach is adaptable to a variety of network topologies. Also, we increase the number of simulation samples in order to check the correctness of our probabilistic models.

As related works, several case studies are performed using PRISM [9]–[11]. For example, Paper [10] deals with

Manuscript received July 19, 2010.
Manuscript revised November 15, 2010.

[†] The authors are with the Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565–0871 Japan.
a) E-mail: t-nagaok@ist.osaka-u.ac.jp

DOI: 10.1587/transinf.E94.D.958

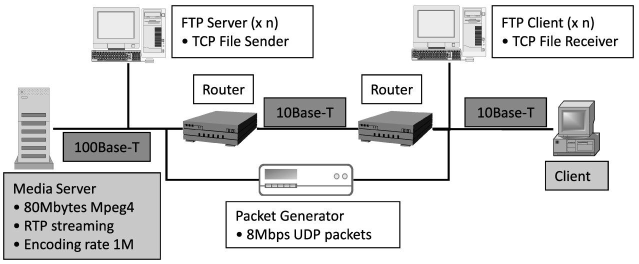


Fig. 1 A configuration of experimental system.

a network protocol. Few works, however, concern the QoS analysis of the whole system. Papers [12] and [13] propose abstraction methods for probabilistic systems based on an abstraction refinement approach. Papers [14] and [15] propose verification approach based on the simulation technique. Paper [12] extracts a number of representative sample paths on a probabilistic model and decides if the model satisfies a given property using such paths. In Paper [15], they model a biomedical sensor network as timed automata, and use the simulation technique to adjust some parameters.

The contribution of the paper includes that we present a technique to guarantee the QoS of real-time distributed systems. Our experimental results show the correctness of our detailed model at least for all of nine scenarios. Also, we can apply probabilistic model checking on the simplified model within realistic time without state explosion. It shows that the proposed method is useful to analyze the network performance. We believe that such analysis is useful for other kind of network analysis.

The rest of the paper is organized as follows. Section 2 gives some backgrounds as preliminaries. Section 3 gives our proposed approach to verify real-time distributed systems without state explosion, and shows an example model described with a PRISM language. Section 4 shows experimental results and gives discussions. Finally Sect.5 concludes the paper.

2. Preliminaries

This section simply describes the probabilistic model checker PRISM as well as network protocols for net streaming.

2.1 Probabilistic Model Checker PRISM

Here, we simply describe an overview of the probabilistic model checker PRISM [5].

A model checking tool usually has two inputs, a model $\mathcal{M}$ and a logical expression p. The model is typically a transition system which represents behavior of the system to check while the logical expression is a temporal logic expression which represents a property to check. The typical output of the model checking tool is whether the logical expression is valid on the model ($\mathcal{M} \models p$). Some model checker outputs a counter example when p is invalid.

The inputs of PRISM include the following three kind of transition systems as a model:

- Discrete-time Markov chains (DTMCs);
- Continuous-time Markov chains (CTMCs); and
- Markov decision processes (MDPs).

Each of three systems is a probabilistic transition system (Markov chain). The inputs of PRISM also include Probabilistic Computation Tree Logic (PCTL) [16] for DTMC and MDP, and Continuous Stochastic Logic (CSL) [17] for CTMC. They are CTL based logics enchanted with probability.

PRISM has several analysis modes: a simulation mode, a numerical analysis mode, and a verification mode. Using the simulation mode, we can observe the behavior of the given model system visually. The numerical analysis mode can evaluate the value of uncertain variable specified with PCTL or CSL based on the model. Such numerical analysis is considered as a kind of parametric model checking [18]. PRISM can draw a graph with several trials of such numeric analysis. The verification mode is like typical model checking except that PRISM cannot output counter examples.

In this paper, we use DTMC's as the model of the network. Here, we describe more precisely on a DTMC. Formally, a DTMC D is a tuple (S, s_init, P, L), where

- S is a set of states ("state space") ;
- $s\_init \in S$ is the initial state;
- $P : S \times S \rightarrow [0, 1]$ is the transition probability matrix where $\Sigma_{s' \in S} P(s, s') = 1$ for all $s \in S$; and
- $L : S \rightarrow 2^{AP}$ is a function labeling states with atomic propositions.

PRISM allows a transition to specify an action and updating expressions on D, where D is a set of variables with finite domains. In other words, a DTMC of PRISM is a kind of an extended automaton with probabilities. Usually, one execution of a transition is translated into a unit time of time elapse (a tick event). Such scheme is known as digital clock view of a DTMC. Using an integer variable (with the upper-bound) explicitly as a clock variable, however, we can also represent a system with discrete time in a DTMC. In this paper, we use the latter scheme to avoid the state explosion problem.

In a PRISM description, a model is composed of a number of modules. Each module is a probabilistic automaton which has some variables and probabilistic transitions. In the PRISM model, those modules can interact with each other. In this paper, the word *module* indicates the module in the PRISM description.

PRISM accepts a reward model in which certain values of rewards are assigned to the states and transitions of the probabilistic model [19]. It allows us to evaluate quantitative properties. For example, if we assign a reward of one to all transitions on the model, we can evaluate an expected number of transitions in the paths to reach a given state from an initial state.

2.2 Protocols for Net-Streaming

Here, we simply summarize typical protocols used in the Internet. Typical protocols used in the Internet have a congestion control mechanism in order to avoid network congestion. For example, TCP (Transmission Control Protocol) uses AIMD (Additive Increase Multiplicative Decrease) type window-flow control as such the mechanism. It controls the data size of sending packets based on the current available bandwidth. Such scheme has an advantage for the correct data transmission. It, however, allows delays, which is not suitable for real-time data transmission. Therefore, RTSP (Real Time Streaming Protocol) is used for real-time application. RTSP is a protocol for the Internet streaming of voice and movies, on TCP/IP network. Famous congestion control mechanisms for RTSP are RAP (Rate Adaptation Protocol) [20] and TEAR (TCP Emulate At Receivers) [21]. Recently, TFRC (TCP-Friendly Rate Control) [22] attracts attention. Hence, this paper models TFRC.

2.2.1 RTSP

RTSP is one of the typical protocols working at end-to-end. RTSP has five states, called SETUP, PLAY, RECORD, PAUSE and TEARDOWN. RTP (Real-time Transport Protocol) is responsible for transmission of stream-data. It determines the throughput of RTP based on a rate control scheme of TFRC using the report message of RTCP (RTP Control Protocol).

2.2.2 TCP Friendly Rate Control TFRC

TFRC is a rate control scheme for fairness between RTP and TCP. It controls the rate in order to avoid bad effects on existing TCP flows in the same network, which increases total effectiveness of the whole network. TFRC controls the rate using a report message of RTSP. The report message contains loss of packets and jitters, which can be estimated via the sequence number of received RTP packets and time stamps, respectively. RFC3448 describes the following formula for determining the throughput:

$$X = \frac{s}{R * \sqrt{2*b*p/3} + (t\_RTO*(3*\sqrt{3*b*p/8}*p*(1+32*p^2)))}$$

where the unit of X is Byte/second. The parameter of the formula is summarized in Table 1. The calculated throughput is a rate with which a RTSP server should send packets considering the network congestion at the time. Therefore weighted average values of the parameters in a short period are applied into the equation. Paper [22] also defines the calculation methods for the parameters. When the value of X is less than the bandwidth, TFRC lets RTSP set the value as throughput.

Table 1 Parameters of the throughput estimation formula.

R[seconds]	Round trip time
p[%]	A packet loss rate
s[Byte]	Packet size
b[number of times]	The number of packets acknowledged by a single TCP acknowledgment
t_RTO[seconds]	A TCP retransmission timeout value

3. Proposed Approach

Probabilistic model checking which can evaluate complicated properties with a high level of confidence is useful for performance evaluation of information systems [4]. However, if we model whole systems with several simultaneous sessions in detail, we cannot avoid the state explosion problem. To avoid the problem, in our approach we model real-time distributed systems in different abstraction levels. Using both of simulation and model checking techniques, we perform qualitative analysis in a stepwise fashion.

For a probabilistic model in a detailed level, we model behaviors of protocols of RTSP, TCP, and UDP in detail, and perform several trials of simulation in order to analyze throughput, packet loss rates, and so on. A simplified model is based on the simulation results. For the simplified model, data transmission which we want to analyze is modeled in detail, while other data transmission is abstracted. For the transmission which we do not concern, we decide transmission rates probabilistically based on the simulation results.

In the rest of this section, we describe the detailed model and the simplified model with a case study of a video data streaming system [23]. In our approach, both of the models are described with the PRISM language.

3.1 Target System

Here, we introduce an example of a real-time distributed system. As the example system, we select a video data streaming system [23] shown in Fig. 1. The system is composed of a pair of a video server and its client, a number of pairs of FTP servers and their clients, and a packet generator, which are connected to each other through routers located at the middle of Fig. 1. The routers are connected through the 10 Base–T Ethernet, which is considered as a bottleneck of packet transmission. In the considering scenario, the video server sends 80 MB of video data with throughput of 1 Mbps using the rate control of TFRC. After 100 seconds from the start of the video streaming, FTP servers and clients start their data transmission through TCP sessions. Also, the packet generator always sends UDP packets with the throughput of 8 Mbps as background noise.

3.2 The Detailed Model

The main component of the detailed model is a queue which buffers packets of a router in a bottleneck link. Behavior such as packet loss and round trip time are based on the state of the queue. Also, for each application in the system, we model behavior of its server in detail, while behavior of its client are abstracted as an operation of dequeue. Time elapsing is controlled discretely with an integer variable. Figure 2 is an abstract outline of the detailed model for the case study of Fig. 1.

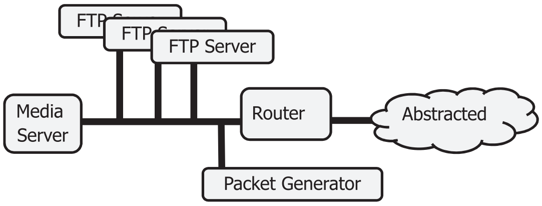


Fig. 2 An abstract outline of the detailed model.

The detailed model is composed of seven modules named *Timer*, *Router*, *MediaServer*, *FTPServer(*×3*)*, *RTTObserver*, *PLRObserver*, while we abstract behaviors of the packet generator as a part of packet transmission behaviors of the module *Router*

3.2.1 Module *Timer*

The module *Timer* manages time elapsing in the detailed model. In this module, we declare an integer variable which represents current time. Time elapsing is based on events such as packet transmission. In the detailed model, each module registers time of occurrence of the next event. When all modules register the time, the module *Timer* performs time elapsing into the latest time of the registered event. After time elapsing, a corresponding module performs the registered event and registers time of the next event again.

The module *Timer* contains two variables and is implemented as eleven lines of code.

3.2.2 Module *Router*

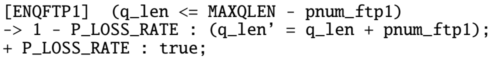
The module *Router* manages buffer control of the router with a queue which buffers transferring packets. In the module, the current queue length is managed with an integer variable. Enqueue and dequeue behaviors are described in the module as operations. Also, regardless of the current queue length, enqueueing packets are dropped with certain probability. In order to construct the module *Router*, we have to specify the maximum length of the queue, a packet transfer rate of the link between the routers, and a constant probability to drop enqueueing packets randomly as its parameters.

In the enqueue operation, if the current queue length becomes larger than the maximum one, the enqueueing packets are dropped (drop tail). The dequeue operation is abstracted; together with the time elapsing operation, a number of packets are output from the queue at a time according to the packet transfer rate of the link.

```
module Router

  q_len : [0..MAXQSIZE] init 0; //The current queue length
  //The history of packet loss intervals
  int_p_loss0 : [0..10000] init 0;
  int_p_loss1 : [0..10000] init 10000;
  int_p_loss2 : [0..10000] init 10000;
  int_p_loss3 : [0..10000] init 10000;
  int_p_loss4 : [0..10000] init 10000;
  int_p_loss5 : [0..10000] init 10000;
  int_p_loss6 : [0..10000] init 10000;
  int_p_loss7 : [0..10000] init 10000;
  int_p_loss8 : [0..10000] init 10000;
  //A flag to observe whether the packet loss
  //                    occurs burstly or not
  p_loss_flag : bool init false;

  //When the queue length does not reach to its maximum
  //Transferring packets are dropped
  //                      with certain probability
  [ENQMS] (q_len <= MAXQLEN - ms_pnum)
    -> 1 - P_LOSS_RATE :
       (q_len’ = q_len + ms_pnum) &
       (p_loss_flag’ = false) &
       (int_p_loss0’ = int_p_loss0 + 1)
     + P_LOSS_RATE :
       (int_p_loss0’ = 0) &
       (int_p_loss1’ = int_p_loss0) &
       (int_p_loss2’ = int_p_loss1) &
       (int_p_loss3’ = int_p_loss2) &
       (int_p_loss4’ = int_p_loss3) &
       (int_p_loss5’ = int_p_loss4) &
       (int_p_loss6’ = int_p_loss5) &
       (int_p_loss7’ = int_p_loss6) &
       (int_p_loss8’ = int_p_loss7) ;

  //When the queue length reaches to its maximum
  [ENQMS] (q_len > MAXQLEN - ms_pnum) & (!p_loss_flag)
    -> (q_len’ = MAXQLEN) & (int_p_loss0’ = 0) &
      (int_p_loss1’ = int_p_loss0 + 1)&
      (int_p_loss2’ = int_p_loss1)&
      (int_p_loss3’ = int_p_loss2)&
      (int_p_loss4’ = int_p_loss3)&
      (int_p_loss5’ = int_p_loss4)&
      (int_p_loss6’ = int_p_loss5)&
      (int_p_loss7’ = int_p_loss6)&
      (int_p_loss8’ = int_p_loss7)&
      (p_loss_flag’ = true);

  // When the packet loss occurs burstly
  // (do not update the history of packet loss intervals)
  [ENQMS] (q_len > MAXQLEN - ms_pnum) & (p_loss_flag)
    -> (q_len’ = MAXQLEN) &
       (p_loss_flag’ = (q_len = MAXQLEN));

  //The ENQUEUE operations for the three FTP sessions
  [ENQFTP1]  (q_len <= MAXQLEN - pnum_ftp1)
    -> 1 - P_LOSS_RATE : (q_len’ = q_len + pnum_ftp1);
     + P_LOSS_RATE : true;

  [ENQFTP1]  (q_len > MAXQLEN - pnum_ftp1)
    -> (q_len’ = MAXQLEN);

  [ENQFTP2] (q_len <= MAXQLEN - pnum_ftp2)
    -> 1 - P_LOSS_RATE : (q_len’ = q_len + pnum_ftp2)
     + P_LOSS_RATE : true;

  [ENQFTP2]  (q_len > MAXQLEN - pnum_ftp2)
    -> (q_len’ = MAXQLEN);

  [ENQFTP3]  (q_len <= MAXQLEN - pnum_ftp3)
    -> 1 - P_LOSS_RATE : (q_len’ = q_len + pnum_ftp3)
     + P_LOSS_RATE : true;

  [ENQFTP3]  (q_len > MAXQLEN - pnum_ftp3)
    -> (q_len’ = MAXQLEN);

  // DEQUEUE is executed together with
  //                      the time elapsing event
  [TIMER] (q_len != 0)
    -> (q_len’ = max(0, q_len - floor(min_lookahead *
                   (RATE_OUT -RATE_PGEN))));

  [TIMER] (q_len = 0) -> true;

endmodule
```

Fig. 3 The module of router described with PRISM language.

Figure 3 shows the module *Router* described with the PRISM language. The module also manages the history of packet loss intervals used for the congestion control. It contains ten variables and is implemented as about 80 lines of code. In Fig. 3, the ten variables are firstly declared. Several actions are defined in CCS like expressions with probabilities. For example, the expression



```
[ENQFTP1]  (q_len <= MAXQLEN - pnum_ftp1)
-> 1 - P_LOSS_RATE : (q_len’ = q_len + pnum_ftp1);
+ P_LOSS_RATE : true;
```

stands for that when the action *ENQFTP1* occurs and ($q\_len \leq MAXQLEN - pnum\_ftp1$) holds, the variables *q_len* is updated to *q_len* + *pnum_ftp1* with probability 1 − *P_LOSS_RATE*, or do nothing with probability *P_LOSS_RATE*.

3.2.3 Module *MediaServer*

The module *MediaServer* manages the transmission of RTSP packets. A packet transmission rate is calculated from the throughput equation defined in [22]. To use the equation, we also have to model round trip time and a packet loss rate of the RTSP session (see Sect. 3.2.5 and 3.2.6). The module *MediaServer* contains six variables and is described with about 37 lines of code.

In our model, the packet transmission behaviors are abstracted as a number of packets are transmitted simultaneously.

3.2.4 Module *FTPServer*

The slow-start and congestion avoidance behaviors of TCP are embedded to the module *FTPServer*. For each connection of the TCP, we declare two integer variables to manage the slow-start threshold and the window size. In total, we declare four variables and the behavior is implemented as about 26 lines of code for each connection.

3.2.5 Module *RTTObserver*

The module *RTTObserver* observes round trip time of RTSP packets. In the module, the round trip time is obtained using physical delay and delay in the router. The delay in the router is calculated as the time to transmit all packets currently buffered in the router. Therefore, we obtain the delay in the router using current queue length and a packet transmission rate of the link.

3.2.6 Module *PLRObserver*

The module *PLRObserver* calculates a packet loss rate of RTSP packets. In the TFRC specification [22], a packet loss rate is calculated using intervals of packet loss. To avoid the loss rate varying rapidly, a history of the packet loss intervals is used. Nine integer variables are declared to manage the history, which are declared in the module *Router*. In the calculation of the loss rate, recent intervals in the history are weighted heavily.

3.3 The Simplified Model

The detailed model described in Sect. 3.2 is too complicated to verify its qualitative properties using probabilistic model checking. Here, we create a simplified model based on simulation results on the detailed model in order to perform model checking. Using the simplified model, we can verify the minimum throughput of the media server.

In the simplified model, behavior of application servers which we do not concern is abstracted. The abstraction is based on the simulation results on the detailed model. In the simulation, we obtain probability distributions of transmission rates for the application servers depending on current queue length. The simplified model decides the packet transmission rate using the distributions to simulate the behaviors of the servers. Figure 4 is an abstract outline of the simplified model. The number of states is a few, thus the module contributes reducing the number of whole states.

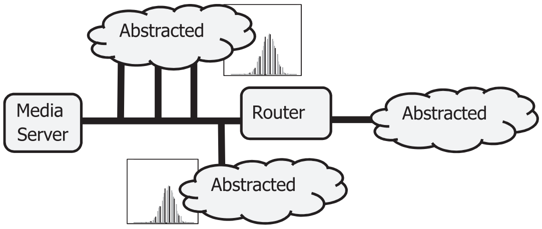


Fig. 4 An abstract outline of the simplified model.

```
rewards "ftp_1th0"
  [CHECK] (prev_q_length >=MAXQSIZE*Q_OCC_LB1) &
          (prev_q_length < MAXQSIZE*Q_OCC_UB1) &
          (ftp_send >= 0) & (ftp_send < 20)
                           : 1; //within 0-80 Kbps
endrewards

rewards "ftp_1th1"
  [CHECK] (prev_q_length >=MAXQSIZE*Q_OCC_LB1) &
          (prev_q_length < MAXQSIZE*Q_OCC_UB1) &
          (ftp_send >= 20) & (ftp_send < 40)
                           : 1; //within 80-160 Kbps
endrewards

rewards "ftp_1th2"
  [CHECK] (prev_q_length >=MAXQSIZE*Q_OCC_LB1) &
          (prev_q_length < MAXQSIZE*Q_OCC_UB1) &
          (ftp_send >= 40) & (ftp_send < 60)
                           : 1; //within 160-240 Kbps
endrewards

rewards "ftp_1th3"
...
```

Fig. 5 A part of reward descriptions for analysis of the distribution.

The simplified model does not have an integer variable to control time elapsing in order to reduce a state space. Instead of using such an integer variable, we assign a certain period into an action transition of the model. Each module transmits a number of packet due to its current transmission rate. In this paper, we let the period be 50 ms.

Here, we describe how to analyze such probability distributions on the detailed model and how to construct the abstracted module from the distributions.

3.3.1 Analysis of the Probability Distributions on the Detailed Model

We use reward descriptions of the PRISM to obtain the probability distributions of transmission rates for application servers. In our detailed model, at every one second in the scenario, we calculate a summation of packet transmission rates for all application servers within the period. If the calculated rate occurs in the range of transmission rates specified by the reward property, we assign the reward one in the reward description. Evaluation of the reward property by the PRISM simulation can generate a histogram of the transmission rates among the extracted paths. Then, we

```
module Other_Servers
  other_rate : [10..31] init 10;
                 // the number of packets in 50 msec]
  [ENQ] (q_length < ceil(85*MAXQSIZE/100)) ->
          0.0000135260 : (other_rate’ = 13) //1040Kbps
        + 0.0000608672 : (other_rate’ = 14) //1120Kbps
        + 0.0004193071 : (other_rate’ = 15) //1200Kbps
        + 0.0022385587 : (other_rate’ = 16) //1280Kbps
        + 0.0094817500 : (other_rate’ = 17) //1360Kbps
        + 0.0281882553 : (other_rate’ = 18) //1440Kbps
        + 0.0671567600 : (other_rate’ = 19) //1520Kbps
        + 0.1294238586 : (other_rate’ = 20) //1600Kbps
        + 0.1955188249 : (other_rate’ = 21) //1680Kbps
        + 0.2156793789 : (other_rate’ = 22) //1760Kbps
        + 0.1728965326 : (other_rate’ = 23) //1840Kbps
        + 0.1040828334 : (other_rate’ = 24) //1920Kbps
        + 0.0497419909 : (other_rate’ = 25) //2000Kbps
        + 0.0188891068 : (other_rate’ = 26) //2080Kbps
        + 0.0051669451 : (other_rate’ = 27) //2160Kbps
        + 0.0008656662 : (other_rate’ = 28) //2240Kbps
        + 0.0001690754 : (other_rate’ = 29) //2320Kbps
        + 0.0000067630 : (other_rate’ = 30); //2400Kbps
  [ENQ] (q_length >=  ceil(MAXQSIZE*85/100))
      & (q_length < ceil(MAXQSIZE*90/100)) ->
          0.0000186290 : (other_rate’ = 12) //960Kbps
        + 0.0000521611 : (other_rate’ = 13) //1040Kbps
        + 0.0002421768 : (other_rate’ = 14) //1120Kbps
        + 0.0009798844 : (other_rate’ = 15) //1200Kbps
        + 0.0047541161 : (other_rate’ = 16) //1280Kbps
        + 0.0163935037 : (other_rate’ = 17) //1360Kbps
        + 0.0452088123 : (other_rate’ = 18) //1440Kbps
        + 0.0971091547 : (other_rate’ = 19) //1520Kbps
        + 0.1661369826 : (other_rate’ = 20) //1600Kbps
        + 0.2123070503 : (other_rate’ = 21) //1680Kbps
        + 0.1995238432 : (other_rate’ = 22) //1760Kbps
        + 0.1393857652 : (other_rate’ = 23) //1840Kbps
        + 0.0729287367 : (other_rate’ = 24) //1920Kbps
        + 0.0315314141 : (other_rate’ = 25) //2000Kbps
        + 0.0102496656 : (other_rate’ = 26) //2080Kbps
        + 0.0025745252 : (other_rate’ = 27) //2160Kbps
        + 0.0005178857 : (other_rate’ = 28) //2240Kbps
        + 0.0000707901 : (other_rate’ = 29) //2320Kbps
        + 0.0000111774 : (other_rate’ = 30) //2400Kbps
        + 0.0000037258 : (other_rate’ = 31); //2480Kbps
  [ENQ] (q_length >=  ceil(MAXQSIZE*90/100))
      & (q_length < ceil(MAXQSIZE*95/100)) ->
          0.0000025649 : (other_rate’ = 11) //880Kbps
        + 0.0000205193 : (other_rate’ = 12) //960Kbps
        + 0.0000718175 : (other_rate’ = 13) //1040Kbps
        + 0.0002872701 : (other_rate’ = 14) //1120Kbps
        + 0.0016005048 : (other_rate’ = 15) //1200Kbps
        + 0.0065046156 : (other_rate’ = 16) //1280Kbps
        + 0.0208270814 : (other_rate’ = 17) //1360Kbps
        + 0.0545941412 : (other_rate’ = 18) //1440Kbps
        + 0.1149516386 : (other_rate’ = 19) //1520Kbps
        + 0.1788974471 : (other_rate’ = 20) //1600Kbps
        + 0.2109511461 : (other_rate’ = 21) //1680Kbps
        + 0.1836220141 : (other_rate’ = 22) //1760Kbps
        + 0.1244212918 : (other_rate’ = 23) //1840Kbps
        + 0.0657463764 : (other_rate’ = 24) //1920Kbps
        + 0.0270623812 : (other_rate’ = 25) //2000Kbps
        + 0.0084539483 : (other_rate’ = 26) //2080Kbps
        + 0.0016877118 : (other_rate’ = 27) //2160Kbps
        + 0.0002744455 : (other_rate’ = 28) //2240Kbps
        + 0.0000230842 : (other_rate’ = 29); //2320Kbps
  [ENQ] (q_length >=  ceil(MAXQSIZE*95/100)) ->
          0.0000074358 : (other_rate’ = 11) //880Kbps
        + 0.0001041011 : (other_rate’ = 12) //960Kbps
        + 0.0002354667 : (other_rate’ = 13) //1040Kbps
        + 0.0007460578 : (other_rate’ = 14) //1120Kbps
        + 0.0026719279 : (other_rate’ = 15) //1200Kbps
        + 0.0078001457 : (other_rate’ = 16) //1280Kbps
        + 0.0243274326 : (other_rate’ = 17) //1360Kbps
        + 0.0614270772 : (other_rate’ = 18) //1440Kbps
        + 0.1240562741 : (other_rate’ = 19) //1520Kbps
        + 0.1859939423 : (other_rate’ = 20) //1600Kbps
        + 0.2053790519 : (other_rate’ = 21) //1680Kbps
        + 0.1726070382 : (other_rate’ = 22) //1760Kbps
        + 0.1167344976 : (other_rate’ = 23) //1840Kbps
        + 0.0625424460 : (other_rate’ = 24) //1920Kbps
        + 0.0255989530 : (other_rate’ = 25) //2000Kbps
        + 0.0078497177 : (other_rate’ = 26) //2080Kbps
        + 0.0016507458 : (other_rate’ = 27) //2160Kbps
        + 0.0002528169 : (other_rate’ = 28) //2240Kbps
        + 0.0000148716 : (other_rate’ = 29); //2320Kbps
endmodule
```

Fig. 6 The abstracted module for four FTP servers.

translate the histogram into the discrete probability distribution. Since the distributions depend on the condition of the queue, we add the condition of latest queue length in the reward description.

Figure 5 shows a part of the reward descriptions in our detailed model. The reward is assigned to the transition labeled with the *CHECK* action. The guard condition for each reward description is composed of the conditions of the queue length and those of the transmission rates. The parameters *Q_OCC_LBn* and *Q_OCC_UBn* ($1 \leq n \leq 4$) stand for lower and upper bounds of queue occupancy rates, respectively. Totally, in our detailed model, we specify 140 of reward properties to obtain the histograms for all conditions of the queue.

3.3.2 Construction of the Abstracted Module

The abstracted module is based on the discrete probability distributions obtained from the results of PRISM simulation. Figure 6 shows the abstracted module obtained in our experiment shown in Sect. 4, which simulates the behaviors of four FTP servers in the example of Sect. 3.1. In the module, there are four transitions labeled with the action *ENQ* with the different guarded conditions involved with the queue length (*q_length*). For each transition, it decides its packet transmission rate (*other_rate*) according to the probability distribution. In the simplified model, the abstracted module interacts with the modules of the media server and the queue.

4. Experiments

We have performed some experiments using our PRISM models described in Sect. 3. We have also modeled the system by NS-2[1], [24] to compare the simulation results. The experiments were performed under an environment of Fedora 13 (64 bit), Intel Core 2 Duo 2.33 GHz, and 2.00 GB of M.M.

In the experiments, we assume packet transmission parameters as follows: the packet size is 500 Byte, the number of packets acknowledged by a single TCP acknowledgment is one, and the TCP retransmission timeout value is $4 \times RTT$ second. We assume that transmitted packets are lost with the probability 0.001.

In this paper, we have performed two experiments.

The first experiment checks the correctness of our detailed model. In the experiment, we performed 1000 trial runs for PRISM and NS-2 simulation, respectively, and compared the simulation results. In this experiment, we consider nine scenarios with respect to the buffer size of routers and the number of the FTP servers. In the scenarios, the buffer sizes are 32, 64, and 128 KB, respectively. Also, the number of the FTP servers are three, four, and five, respectively.

In the second experiment, we performed about 10000 trial runs for PRISM simulation, and created a more simplified PRISM model based on the simulation results. As a target scenario, we selected a scenario of 64 KB of buffer and four FTP connections. Using the simplified model, we verified the minimum throughput of the RTP session.

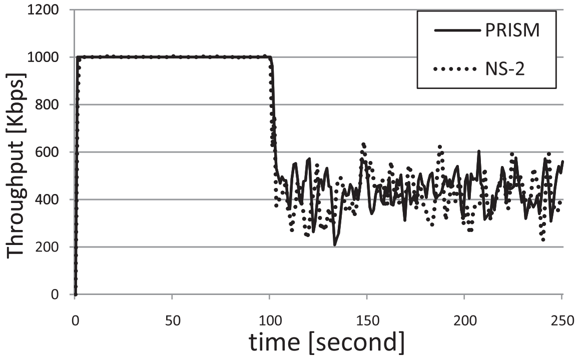


Fig. 7 Comparison of the throughput.

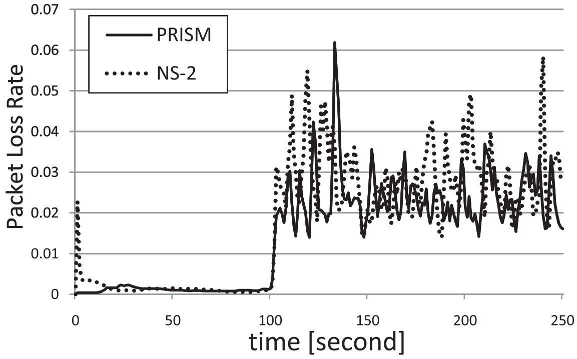


Fig. 8 Comparison of packet loss rates.

4.1 Analysis of the Correctness

Before analyzing the 1000 trials of simulation, we extracted one sample from the simulation results by PRISM and NS-2, respectively. Figure 7 and Fig. 8 represent measured throughput and packet loss rates, respectively, in the scenario of 64 KB of the buffer and four FTP connections. Throughput in the graph means the average throughput within one second, and a packet loss rate means a calculated value at the time as defined in [22]. In the scenario of the example, file transmission starts after 100 seconds from the start of the RTP session, and this causes the network congestion. Consequently, the throughput of the RTP session goes down and the packet loss rate of it comes up. The simulation results of Fig. 7 and Fig. 8 show that our PRISM model and NS-2 model behave similarly even if the network congestion occurs.

To analyze the correctness of our PRISM model in detail, we compare the average, variance, minimum and maximum of throughput of the media server in the 1000 of runs measured by PRISM and NS-2. Tables 2, 3 and 4 represent the analyzed throughput in the period of congestion (after 120 seconds in the simulation scenario). The row of *Size* stands for the buffer size of the router. Also the rows of *Max*, *Min*, *Ave* and *Var* represent the maximum (Kbps), minimum (Kbps), average (Kbps), and variance of throughput, respectively.

Table 2 Summary of the analyzed data (3 FTP servers).

Size	32 KB		64 KB		128 KB	
Model	NS-2	PRISM	NS-2	PRISM	NS-2	PRISM
Max	1000	880	960	920	972	956
Min	44	24	128	132	224	128
Ave	536	523	515	530	535	513
Var	1.55E+04	1.20E+04	1.02E+04	1.03E+04	8.10E+03	1.20E+04

Table 3 Summary of the analyzed data (4 FTP servers).

Size	32 KB		64 KB		128 KB	
Model	NS-2	PRISM	NS-2	PRISM	NS-2	PRISM
Max	992	796	856	832	768	800
Min	20	4	76	56	100	128
Ave	387	368	399	421	399	399
Var	1.37E+04	7.77E+03	7.82E+03	7.14E+03	5.09E+03	5.92E+03

Table 4 Summary of the analyzed data (5 FTP servers).

Size	32 KB		64 KB		128 KB	
Model	NS-2	PRISM	NS-2	PRISM	NS-2	PRISM
Max	816	644	688	660	664	680
Min	4	4	36	40	84	100
Ave	289	270	317	332	317	330
Var	1.15E+04	5.63E+03	6.43E+03	5.35E+03	3.77E+03	3.94E+03

Tables 2, 3, and 4 show that the behavior of our detailed model is similar with that of the NS-2 model for all scenarios. Also we have analyzed packet loss rates and RTT as well. The results also show our detailed model behaves similarly to the NS-2 model. In the cases of the buffer size 32 KB, however, we can see the difference of the maximum throughput between the detailed model and NS-2. We think one of the reasons is that we strongly abstract a packet sending mechanism in the PRISM model, that is, when the packet transmission rate is high, our model transmits a number of packets at a time. When the buffer size is small, transmitted packets tend to be lost because of such abstraction. We think this causes the differences of behaviors between the PRISM model and NS-2 one.

For one trial run of the PRISM simulation, it takes 2.5 seconds averagely, while it takes 34.1 seconds in the simulation of NS-2.

4.2 Verification Results for the Simplified Model

Here, we verify the minimum throughput that the media server may provide in the worst case. In the verification, we use a simplified model based on the simulation results on the detailed PRISM model. The simulation results should contain discrete probability distributions of the throughput of FTP servers and the packet generator. Since the packet generator generates UDP packets at the same rate in the example, we can decide the transmission rate to be the same rate.

The discrete probability distributions of the total throughput of the four FTP servers are shown in Fig. 9, where the buffer size is 64 KB and the number of the FTP is four. The charts are divided with respect to values of buffer occupancy. It takes about 173 minutes to perform 10000 trials of PRISM simulation.

Based on the results of Fig. 9, we create the simplified

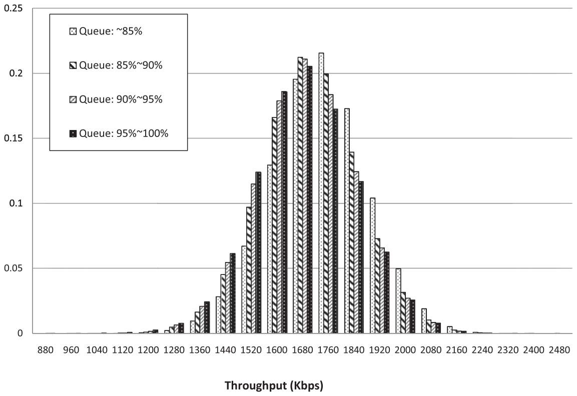


Fig. 9 The discrete probability distribution of throughput of the FTP servers.

model. The simplified model is described with 136 lines of code and seven variables. In order to reduce a state space, we have reduced a range of integer variables for the control of packet loss intervals.

The verification property for checking the minimum throughput is given as follows; $P_{>0}$ [F *measure* & *throughput* $\leq x$]. This property means the probability of *throughput* being the value of x is greater than 0, where *throughput* is a variable to measure throughput of the RTP session within one second and *measure* is a boolean variable which represents timing to measure the throughput. In order to evaluate the value of throughput within one second, we let *measure* become true periodically for every one second (20 steps) in the scenario. The verification is performed with varying the value of x from 0. The minimum throughput is defined as a minimum value of x such that the result of model checking becomes true. In the experiment, we have performed model checking with varying x from 0 to 56 which is the minimum throughput obtained by simulation on the detailed model.

Model checking on the simplified model outputs the minimum throughput 20 Kbps. The number of states constructed by PRISM is 10885476, and it takes 940 seconds for model checking. We can see that the obtained minimum throughput doesn't contradict with the simulation results shown in Table 3.

4.3 Discussion

In the simulation on the detailed model, we have obtained similar results with NS-2. We conclude that we have modeled correctly the behavior of real-time distributed systems. Also, the execution time of PRISM simulation for one trial is shorter than that of NS-2. We think that this is because NS-2 implements behavior of protocols definitely while our detailed model has some abstracted behavior. From the experimental results, we expect that we can use PRISM as a network simulator for the real-time distributed systems.

In the verification on the simplified model, it has taken about 173 minutes for simulation and about 15 minutes for model checking. We think that we can verify the property within the realistic time. For the state space, we have constructed about ten millions of states, though we reduce the space by some abstraction. In order to analyze other qualitative properties or apply our technique into more complicated systems, we have to apply other abstraction techniques to reduce the state spaces as a future work.

For validity, we did not fully show the validity of our simplified model by the experiment. Therefore, we cannot say the results of probabilistic model checking are reliable. We believe that, however, as reported in Paper [4] there are many works in which they apply probabilistic model checking to performance evaluation of information systems, and our hybrid approach is useful to reduce verification costs. We have to show the validity of our simplified model by performing other experiments in the future.

5. Conclusion

This paper presents a hybrid evaluation method for a real-time distributed system based on the probabilistic model checking technique and simulation. In our approach, we perform stepwise analysis using probabilistic models of target systems in different abstract levels (detailed model and simplified model). To validate the correctness of our model, we model it in a model for the well-known network simulator NS-2, and give the comparison of their simulation results. The comparison shows that the result of PRISM simulation is very similar to that of NS-2. It shows that the proposed approach is useful to analyze the network performance. We believe that such analysis is useful for other kind of network analysis.

The future works include validation of our simplified model, and also automatic derivation of the simplified model suitable for model checking analysis. Many abstraction techniques are proposed for model checking. We want to apply such techniques to the process.

Acknowledgments

This work is being conducted as a part of Stage Project, the Development of Next Generation IT Infrastructure, supported by Ministry of Education, Culture, Sports, Science and Technology, as well as Grant-in-Aid for Scientific Research C(21500036), as well as grant from The Telecommunications Advancement Foundation.

References

[1] "The Network Simulator - ns-2," available at http://www.isi.edu/nsnam/ns/
[2] E.J. Kim, K.H. Yum, and C.R. Das, "Calculation of deadline missing probability in a QoS capable cluster interconnect," Proc. IEEE Int. Symp. on Network Computing and Applications (NCA'01), p.36, Oct. 2001.
[3] E.M. Clarke, O. Grumberg, and D.A. Peled, eds., Model Checking, MIT Press, 1999.
[4] C. Baier, B.R. Haverkort, H. Hermanns, and J.P. Katoen, "Performance evaluation and model checking join forces," Commun. ACM, vol.53, no.9, pp.76–85, Sept. 2010.

[5] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A tool for automatic verification of probabilistic systems," Proc. 12th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06), vol.3920 of LNCS, pp.441–444, March 2006.

[6] A. Ito, T. Nagaoka, K. Okano, and S. Kusumoto, "Verification for the real-time network systems with the probabilistic model checker and its comparison with the network simulator Ns-2," IEICE Technical Report, SS2009-18, Aug. 2009.

[7] T. Nagaoka, A. Ito, K. Okano, and S. Kusumoto, "QoS evaluation for real-time distributed systems using the probabilistic model checker PRISM," Proc. Int. Workshop on Informatics 2009, pp.60–66, Sept. 2009.

[8] E. Nagai, K. Okano, and S. Kusumoto, "Performance verification considering the network delay for real-time distributed systems with the probabilistic model checker PRISM," IEICE Technical Report, SS2005-86, Feb. 2006.

[9] M. Kwiatkowska, G. Norman, and J. Sproston, "Performance analysis of probabilistic timed automata using digital clocks," Formal Methods in System Design, vol.29, no.1, pp.33–78, Aug. 2006.

[10] M. Kwiatkowska, G. Norman, and J. Sproston, "Probabilistic model checking of the IEEE 802.11 wireless local area network protocol," Proc. 2nd Joint Int. Workshop on Process Algebra and Performance Modelling and Probabilistic Methods in Verification (PAPM/PROBMIV'02), vol.2389 of LNCS, pp.169–187, July 2002.

[11] M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang, "Symbolic model checking for probabilistic timed automata," Inf. Comput., vol.205, no.7, pp.1027–1077, July 2007.

[12] M. Kattenbelt, M. Kwiatowska, G. Norman, and D. Parker, "A game-based abstraction-refinement framework for Markov decision processes," Int. Journal on Formal methods in System Design, vol.36, no.3, pp.246–280, Sept. 2010.

[13] A. Morimoto, R. Komagata, and S. Yamane, "Probabilistic timed CEGAR," IEICE Technical Report, CST2009-5, June 2009.

[14] E.M. Clarke, A. Donzé, and A. Legay, "On simulation-based probabilistic model checking of mixed-analog circuits," Formal Methods in System Design, vol.36, no.2, pp.97–113, June 2010.

[15] S. Tschirner, L. Xuedong, and Y. Yi, "Model-based validation of QoS properties of biomedical sensor networks," Proc. 8th ACM Int. Conf. on Embedded software, pp.69–78, Oct. 2008.

[16] H. Hansson and B. Jonsson, "A logic for reasoning about time and probability," Formal Aspects of Computing, vol.6, no.5, pp.512–535, Sept. 1994.

[17] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying continuous time Markov chains," Proc. 8th Int. Conf. on Computer Aided Verification (CAV'96), vol.1102 of LNCS, pp.269–276, July 1996.

[18] R. Alur, T.A. Henzinger, and M.Y. Vardi, "Parametric real-time reasoning," Proc. 25th ACM Annual Symp. on the Theory of Computing (STOC'93), pp.592–601, May 1993.

[19] "PRISM Manual," http://www.prismmodelchecker.org/manual/

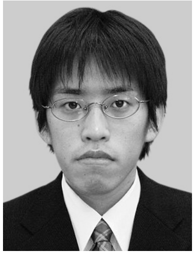
[20] R. Rejaie, M. Handley, and D. Estrin, "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet," Proc. IEEE INFOCOM 1999, vol.3, pp.1337–1345, March 1999.

[21] I. Rhee, V. Ozdemir, and Y. Yi, "TEAR: TCP emulation at receivers – Flow control for multimedia streaming," Technical Report, NCSU, 2000.

[22] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "TCP friendly rate control (TFRC): Protocol specification," RFC 3448 (Proposed Standard), Jan. 2003, Obsoleted by RFC 5348.

[23] Y. Taniguchi, A. Ueoka, N. Wakamiya, M. Murata, and F. Noda, "Implementation and evaluation of proxy caching system for MPEG-4 video streaming with quality adjustment mechanism," Proc. 5th Association of East Asian Research Universities Workshop on Web Technology, pp.27–34, Oct. 2003.

[24] D. Mahrenholz and S. Ivanov, "Real-time network emulation with ns-2," Proc. 8th IEEE Int. Symp. on Distributed Simulation and Real-Time Applications, pp.29–36, Oct. 2004.



Takeshi Nagaoka received the M.I. degree in Computer Science from Osaka University in 2007. He currently belongs in a doctoral course. His research interests include abstraction techniques in model checking, especially a timed automaton and a probabilistic timed automaton.

Akihiko Ito received the BE and MI degrees in Computers Sciences from Hiroshima University and Osaka University in 2008 and 2010, respectively. His research interests include model checking, especially timed automaton and probabilistic timed automaton.

Kozo Okano received the BE, ME, and Ph.D degrees in Information and Computer Sciences from Osaka University, in 1990, 1992, and 1995, respectively. Since 2002 he has been an associate professor in the Graduate School of Information Science and Technology, Osaka University. In 2002, he was a visiting researcher of the Department of Computer Science, University of Kent at Canterbury. In 2003, he was a visiting lecturer at the School of Computer Science, University of Birmingham. His current research interests include formal methods for software and information system design. He is a member of IEEE and IPS of Japan.

Shinji Kusumoto received the BE, ME, and DE degrees in information and computer sciences from Osaka University in 1988, 1990, and 1993, respectively. He is currently a professor in the Graduate School of Information Science and Technology at Osaka University. His research interests include software metrics and software quality assurance technique. He is a member of the IEEE, the IEEE Computer Society, IPSJ, and JFPUG.