2009 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems

Modeling and Analysis of Probabilistic Timed Systems

Abhishek Dubey [†] Derek Riley[†] Sherif Abdelwahed[‡] Ted Bapty[†]

[†] Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

[‡] Electrical Engineering and Computer Science, Mississippi State University

Abstract

Probabilistic models are useful for analyzing systems which operate under the presence of uncertainty. In this paper, we present a technique for verifying safety and liveness properties for probabilistic timed automata. The proposed technique is an extension of a technique used to verify stochastic hybrid automata using an approximation with Markov Decision Processes. A case study for CSMA/CD protocol has been used to show case the methodology used in our technique.

1. Introduction

Probabilistic models are useful for analyzing systems which operate under the presence of uncertainty. These systems can be modeled using non-determinism; however, often more information is known about the probabilities associated with certain aspects of the system. Also, many systems such as slot machines and communication networks are specifically designed to execute probabilistic behavior to incorporate fairness into their decision making choices.

Soft real-time systems exhibiting probabilistic behavior can be analyzed using these probabilistic models to more realistically study their behaviors. Alternatively, hard real-time systems may also exhibit probabilistic behavior; however, the hard deadlines of the system do not allow for any probability of failure. Therefore, probabilistic analysis is most beneficial for soft real-time systems.

In the past, many communication protocols e.g. Carrier Sense Multiple Access and Collision Detection, IEEE 1394 firewire protocol, Bluetooth device discovery [12], as well as biological systems have been modeled and analyzed as probabilistic models [13]. In the case of communication protocols the results from these analysis' can help lead the designers toward better solutions which would give a higher probability of success. On the other hand the analysis from biological systems can help researchers develop a better understanding of the complex physiological dynamics of the systems they study. Probabilistic models are also useful in modeling effect of possible failures [22].

In this work, we present a technique for modeling and analyzing probabilistic timed systems by approximating them as a Markov Decision Process (MDP) and analyzing them with the Value Iteration (VI) technique [20] We will discuss the application of this technique using a case study on a modified CSMA/CD protocol. This analysis can help designers choose the parameters for the protocol which most likely will meet their specifications. The outline for the rest of the paper is as follows: Section 2 gives a background and related research; Section 3 explains the model that we use for probabilistic timed systems. In Section 4 and Section 5 we describe the various kinds of analyses that we can perform on this model and the technique we use to perform these analyses. We conclude the paper by demonstrating our approach in Section 6.

2. Modeling Formalisms for Probabilistic Systems

We can classify probabilistic systems into two groups, *purely probabilistic* and *generalized probabilistic* [23]. A system is called purely probabilistic if, for every state of the system, there is exactly one and only one set of probability distribution over all available transitions. Non-determinism occurs, when (a) there are multiple possible transition without an associated probability distribution, or (b) there are multiple probability distributions defined over the same set of available transitions. In [19], systems which do not have such non-determinism are modeled as stochastic automaton with only one possible transition matrix without any external actions.

For non-deterministic systems (generalized probabilistic systems) with multiple probability distributions defined over a set of transitions as external agent referred to as adversaries [18], strategies [6], schedulers [24] or external actions [19] decide the chosen probabilistic distribution.

A basic model used in describing behaviors of probabilistic systems is a Markov Chain. It is a simple "random walk" where the state space is represented as vertices of a graph and the transition can involve a movement to any ad-

978-0-7695-3602-6/09 $25.00 © 2009 IEEE
DOI 10.1109/ECBS.2009.44

jacent neighbor such that the movement in current time step is independent of all previous movements. This assumption of conditional independence from all previous states except the most recent one is also called "Markovian assumption". Most probabilistic models are enrichments of this Markovian chain model [21].

The finite-state discrete time Markov Chain (DTMC) [17] is commonly used to model pure probabilistic systems. DTMC satisfies the Markovian assumption i.e. it is assumed that the probability of reaching a certain state in next time step only depends upon the present state and not the past states. Thus, the probability of system taking any path $\pi = s_0, s_1 \cdots s_n$, is equal to $\prod_1^{n-1} P(s_{i+1}|s_i)$, where s_i is a state of the system, and $P(s_{i+1}|s_i)$ is the probability that the next state will be s_{i+1}, given that the current state is s_i. Thus, given an initial state and an assignment of propositional labels for each state one can solve the probability of reaching a certain state and satisfying some propositional logic formula by summing the probability of all paths between those two states.

A related model is continuous time markov chain model (CTMC). It associates a probability distribution function with a continuous range of time. In a CTMC, state transitions may occur at any time such that time between transitions is exponentially distributed i.e. probability of a current transition does not depend on previous transitions [21].

A Markov Decision Process (MDP) [20], extends the basic notion of a DTMC by allowing multiple probabilistic distributions specifying the probability of state transitions. This system is very commonly used in artificial intelligence to choose an optimum policy for reaching a desired goal state given a set of actions, which have different transition probability distribution associated with them. Such MDP also specify a reward model for choosing an action and a cost for taking an action.

MDP is similar to the Probabilistic Nondeterministic system (PNS) of Bianco and de Alfaro [6]. PNS allows under specification of a transition probability. Under specification here means that the PNS model gives the flexibility of leaving some transition probabilities unspecified. However, it is straight forward to map the non-deterministic under specified probabilistic transition of a PNS as an extra probabilistic distribution of a MDP in which only that transition has a unity probability while all the other possible transitions are given a zero probability. The semantics of PNS require a strategy to pick a probability distribution and then define state transitions according to the chosen distribution. This model has been shown to be able to represent n asynchronous purely probabilistic systems running concurrently. It should be noticed that due to non-determinism, now only a max and min probability of reaching a state can be specified.

Generally, PNS are not used to model timed systems, unless it is assumed that every transition takes the same amount of time. This notion is similar to the notion of RT-CTL model checking for Kripke structures in NuSMV [8]. However, in real systems where different events take different time, PNS cannot be used.

In [11], timed probabilistic non-deterministic systems (TPNS) model was introduced. A TPNS model is similar to PNS and have non-deterministic actions chosen by some adversary associated with every state. These actions decide a particular probability distribution for the transition to the next state. The difference, however, between a TPNS and PNS is the cost associated with any choice of action. The cost of an action, which is either 0 or 1 is directly related to the time spent in the present state before executing that action. Similar to a PNS, only maximum and minimum reachability probability for a certain state is specified.

In order to specify the properties of these systems, probabilistic logics are used. Probabilistic real-time computational tree logic PCTL [14] is used for expressing real-time and probability in systems. This logic was extended to PCTL* in [3], much the same as CTL* extends CTL [15]. PCTL* differs from CTL* in that the path quantifiers A and E are absent and instead a probability operator $[\psi]_{\sqsupseteq p}$ is specified for the path formula ψ. This operator is interpreted as the probability of satisfaction of the path formula ψ is less ($\sqsupseteq = \leq$) or more ($\sqsupseteq = \geq$) than p. The semantics of PCTL and PCTL* is defined with respect to finite Markov chains. It should be noted that in PNS models a PCTL* formula $[\psi]_{\sqsupseteq p}$ is compared against the maximum and minimum probability computed for a particular path.

Baier and others defined a probabilistic branching time logic (PBTL) in [4] that extends the probabilistic logic for PNS models. This logic adds the concept of fairness to PCTL*. Thus any conditions which lead to an unfair path are not considered while evaluating the satisfaction of a PBTL formula.

The underlying semantic model for all these logics is either a DTMC or MDP. There have been significant development in probabilistic model checking and a number of tools such as PRISM [17] and Mobiüs [9] have been developed. However, unlike PRISM in which systems have to be specified as a concurrent DTMCs, or concurrent MDPs, or CTMCs, Mobiüs allows a rich set of modeling formalisms like stochastic petri nets [7] for checking the reliability of a system.

The "real-time" probabilistic model, also called as probabilistic timed automaton (PTA) was first introduced in [1]. This paper described a probabilistic logic that can be used to specify interesting temporal properties of the probabilistic system. In this paper, we show how a probabilistic timed automaton can be converted to an equivalent MDP and also show how we can use the Value Iteration algorithm [20] to analyze reachability properties.

3. Probabilistic Timed Automaton

A timed automaton [2] is a 6-tuple TA = $< \Sigma, S, L, S_0, X, I, T >$ such that

- Σ is a finite set of alphabets, which the TA can accept.
- S is a finite set of locations.
- L: $S \rightarrow 2^{AP}$ is a labeling function, where AP is the set of all atomic propositions
- $S_0 \subseteq S$ is a set of initial locations.
- X is a finite set of clocks.
- $I : S \rightarrow C_X$ is a mapping called location invariant. C_X is the set of clock constraints over X defined in BNF grammar by $\alpha ::= x \prec c | \neg\alpha | \alpha \wedge \alpha$, where $x \in X$ is a clock, $\alpha \in C_X$, $\prec \in \{<, \leq\}$, and c is a rational number.
- $T \subseteq S \times \Sigma \times C_X \times 2^X \times S$ is a set of transitions. The 5-tuple $< s, \sigma, \psi, \lambda, s' >$ corresponds to a transition from location s to s' via an alphabet σ, a clock constraint ψ specifies when the transition will be enabled and $\lambda \subseteq X$ is the set of clocks whose value will be reset to 0.

Probabilistic Timed Automata (PTA) [18] are an extension to Timed Automata (TA) which includes probabilistic transitions between the states of the system. The formal definition for a PTA G is as follows:

$G = (\Sigma, S, L, S_0, X, I, T, Prob, PDF, G_s)$:

- Σ is a finite set of alphabets, which the PTA can accept.
- S a finite set S of locations, where s_0 is the initial location.
- L: $S \rightarrow 2^{AP}$ is a labeling function, where AP is the set of all atomic propositions
- A finite set X of clocks.
- $S_0 \subseteq S$ is a set of initial locations.
- $I : S \rightarrow C_X$ is a mapping called location invariant. C_X is the set of clock constraints over X defined in BNF grammar by $\alpha ::= x \prec c | \neg\alpha | \alpha \wedge \alpha$, where $x \in X$ is a clock, $\alpha \in C_X$, $\prec \in \{<, \leq\}$, and c is a rational number.
- $T \subseteq S \times \Sigma \times 2^X \times S$ is a set of transitions. The 5-tuple $< s, \sigma, \lambda, s' >$ corresponds to a transition from location s to s' via an alphabet σ, and $\lambda \subseteq X$ is the set of clocks whose value will be reset to 0. Note the difference from the TA definition. Here the guard condition is not specified with the transition. Let $src : T \rightarrow S$ be a map that defines the source location for a transition.

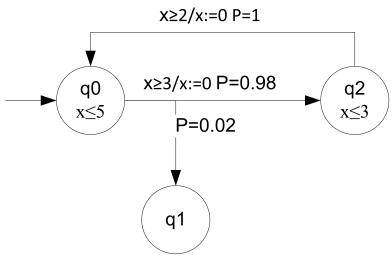

Figure 1. An example of a probabilistic timed automaton.

- A function $Prob$: $S \rightarrow 2^{\mu(T)}$, assigns to each location a finite non-empty set of discrete probability distributions $\mu : T \rightarrow [0, 1] \cup \Phi$ on T such that $(\forall s \in S) \sum_{t \in T \wedge s = src(t)} \mu(t) \leq 1$. Φ is the null set and models the underspecification of probability i.e. one can choose to not specify the probability for a transition. Generally, we deal with systems that only have one probability distribution defined over all state i.e $(\forall s \in S)|Prob(s)| = 1$ (purely probabilistic). However, for other systems which are generalized we depend on some other random variable to pick a probability distribution $\mu \in Prob(s)$ - as described in [19].
- A family (sequence) of functions $< G_s >_{s \in S}$, $G_s : Prob(s) \rightarrow C_X$ assigns to each $p \in prob(s)$ a guard. These guard conditions serve the function of enabling a probability distribution, which then enables certain transitions.

3.1. Semantics of Probabilistic Timed Automaton

Fig. 1 shows an example of probabilistic timed automaton. Notice that the transition from q0 state can either go to the q1 state or the q2 state.

The state of a probabilistic timed automaton is a pair of the current evaluation of clock variables and the current location, $Q = (s, v)$, where $s \in S$ and $v : X \rightarrow \mathbb{R}^+$ is the clock value map, assigning each clock a positive real value. At any time, all clocks increase with a uniform unit rate, which, along with events, enable transitions from one state of the timed automaton to another. Since there are an infinite number of possible clock evaluations, the state space of a timed automaton is infinite. The transition graph over this state space, $A =< \Sigma, Q, Q_0, R >$, is used to describe the semantics associated with this model. The initial state of A, Q_0 is given by $\{(q, v) | q \in S_0 \wedge \forall x \in X(v(x) = 0)\}$.

The transition relation R is composed of two types of transitions: delay transitions caused by the passage of time,

and action transitions, which lead to a change in location of a timed automaton. Before proceeding further with transitions, it is necessary to first define some notation.

Let us define $v + d$ to be a clock assignment map, which increases the value of each clock $x \in X$ to $v(x) + d$. For $\lambda \subseteq X$ we introduce $v[\lambda := 0]$ to be the clock assignment that maps each clock $y \in \lambda$ to 0, but keeps the value of all clocks $x \in X - \lambda$ same. Using these notations, we can define delay and action transitions as follows:

- *Delay Transitions* refer to passage of time while staying in the same location. They are written as $(s, v) \xrightarrow{d} (s, v + d)$. The necessary condition is $v \in I(s)$ and $v + d \in I(s)$
- *Action Transitions* refer to occurrences of a transitions from one location to another location. Given an alphabet σ, an action transition for a probabilistic timed automaton is composed of two steps:
  1. Choose a probability distribution $p \in prob(s)$ such that the current clock valuation v satisfies the guard $G_s(p)$ i.e. $v \in G_s(p)$. This choice is some
  2. Make a probabilistic transition to a state s' according to p; that is for any state $s' \in S$ and $\lambda \subseteq X$, the probability that location will change to s' and clocks λ will be reset to 0 is given by $p(t)$, where $t = (s, \sigma, \lambda, s')$ is a transition.

Probabilistic Timed Automaton differ from TA in the fact that they allow, but do not require probabilities on transitions. These probabilities must be assigned to states of the TA which only allow discrete, not timed transitions. A PTA must be very carefully defined because the transitions of a TA which allow both timed and discrete transitions generate inherent non-determinism which cannot be resolved through simple probability assignment. Therefore, our verification technique ignores probabilities assigned to these types of transitions and keeps them as non-deterministic so the results of the verification do not make limiting assumptions which the user is not aware of. This problem can be avoided if the TA is designed in such a way to limit or eliminate the use of these special non-deterministic choices.

Networked PTA: Usually, a system is composed of several sub-systems, each of which can be modeled as a timed automaton. Therefore, for modeling of the complete system, we will have to consider the parallel composition of a network of timed automatons [5, 10, 25].

A network of timed automatons is a parallel composition of several timed automatons [10]. Each timed automaton can synchronize with any other timed automaton by using input events and output actions. For this purpose, we assume the alphabet set Σ to consist of symbols for input events denoted by $\sigma?$ and output actions $\sigma!$ and internal events τ. Networked probabilistic timed automaton are defined similarly.

The semantics of network-timed automatons are also given in terms of transition graphs. A state of a network is defined as a pair $(\vec{s}, v)$, where $\vec{s}$ denotes a vector of all the current locations of the network, one for each timed automaton, and v is the clock assignment map for all the clocks of the network. The rules for delay transitions are the same as those for a single timed automaton. However, the action transitions are composed of internal actions and external actions.

An internal action transition is a transition, which happens in any one timed automaton of the network, independent of other timed automatons. On the other hand, an external action transition is a synchronous transition between two timed automatons. For such a transition, one timed automaton produces an output event on its transition leading to a change in its location (denoted as $a!$), while the other timed automaton consumes that event (denoted as $a?$) and takes the transitions leading to a change in its location. An external action transition cannot happen if any of the timed automatons cannot synchronize.

For a networked probabilistic timed automaton, the probability of synchronized transitions is obtained by multiplying the probability of individual transitions.

Before going on to next section, let us assume the existence of an operation $unprob(.)$ that generates an equivalent timed automaton from a given probabilistic timed automaton. For purely probabilistic systems, one can find a bisimilar TA by simply equating all transition probabilities to one. That is system is turned into a non-deterministic system from a probabilistic system. For a system that has multiple probability distributions, we have to first convert it into a purely probabilistic system by invoking an adversary [18] that chooses a probability distribution for all source states. We can then use the unprob operator.

Definition 1 (Probabilistically Bisimilar) *A PTA $G = (\Sigma, S, L, S_0, X, I, T, Prob, G_s)$ and a TA defined over same locations,clock variables, initial locations, Invariants and alphabets TA=$< \Sigma, S, L, S_0, X, I, T >$ are probabilistically bisimilar iff for all probabilistic transitions of a PTA with a given guard condition an equivalent transition of timed automaton with same guard condition exists i.e.*

$$(\exists p \in Prob(s))(p(s, \sigma, \lambda, s') \neq 0) \quad (1)$$
$$\Leftrightarrow (\exists t \in TA.T)(t = (s, \sigma, G_s(p), \lambda, s'))$$

4. Analysis Types

Our proposed verification method allows for several different types of verification including reachability, safety, and bounded time reachability.

4.1. Reachability

The result of reachability analysis describes the probability that a certain state will transition to a set of states initially marked as reachable while not transitioning to another set of states initially marked as unreachable. The reachable states can represent states such as done, and the unreachable states can represent states such as error. The resulting reachability probability will then describe the probability that a state will transition to a done state without reaching an error state i.e. given a set of target locations $\phi \subseteq 2^{AP}$, and a set of unsafe locations ψ, find the conditional probability $P(E\Diamond\phi \wedge \ not\ A\Box\psi|S_0)$, where TCTL formula $E\Diamond\phi$ is true if the predicate logic formula ϕ is eventually satisfied on any execution path of the system. S_0 is the initial location.

4.2. Safety

The result of safety analysis on a system describes the probability that a state will never reach a set of bad states. These marked states must be identified a priori and can describe states such as error or completed. Safety analysis is performed as a negation of reachability check. For example if ψ is the set of unsafe location, then safety probability is $1 - P(E\Diamond\psi|S_0)$.

4.3. Bounded Time Safety/Reachability

Bounded time reachability or safety can be determined for PTA by using reachability or safety analysis on a modified version of the PTA. To add the bounded time analysis, it is required that the user add an additional clock to the system which marks the total time of the system and is never reset. The user must also add a new error state to the system which is transitioned to by any state of the system after the bounded time is reached. The analysis method is similar to the reachability or safety analysis described above except that the new error state is marked as the reachable or unsafe set. The result of this analysis will tell the user the probability that the initial state will transition to the error state (run out of time) before transitioning to a safe state.

5. Verification Technique

Figure 2 illustrates the control flow of our approach. Given a PTA, we first generate an equivalent TA by using the unprob(.) operator. The probability distributions associated with all guard conditions and transitions are stored in a separate file as a lookup table. This lookup table stores all probability distributions and corresponding probabilities stored for all transitions. A transition is identified using the source and destination location pair. Next step involves generation of the equivalency graph for the timed automaton TA by using KRONOS tool [25]. The equivalency graph contains a set of symbolic states $< s, \mathscr{Z} >$, where $s \in S$ is a location and $\mathscr{Z} \subseteq C_x$ is set of clock valuations such that $\mathscr{Z} \subseteq I(s)$: it represents all states (s, v) such that $v \models \mathscr{Z}$. It also contains the discrete transitions between these states. This graph is called equivalency graph because it is bisimilar to the given TA. More details can be founds in [25].

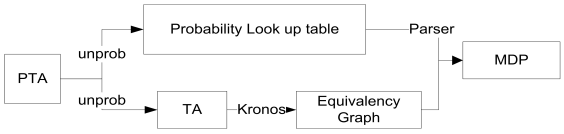

Figure 2. Algorithmic Flow.

An equivalency graph is effectively a finite state machine over the set of all states of timed automaton and the alphabets Σ. Once the equivalency graph is generated, a parser program uses that file, along with another file describing the probabilities of the system to create the PTA representation. This representation is equivalent to a Markov Decision Process (MDP), which is then analyzed using a technique called the value iteration method. We will discuss this further in section 5.2.

Lemma 1 *Behaviors generated by the generated markov decision process are probabilistically bisimilar to the behaviors generated by the orginal probabilistic timed automaton.*

Proof 1 *Since the TA is probabilistically bisimilar to the PTA and the equivalency graph is bisimilar to the TA, by associativity we can maintain that the markov decision process is probabilistically bisimilar to the original PTA.*

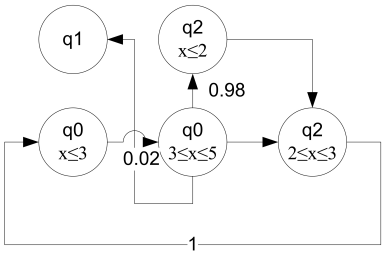

Figure 3. The equivalency graph with probabilities for PTA shown in fig. 1.

A Markov Decision Process requires that all transition probabilities between all states are well defined. Therefore,

the parsing program which translates the reachability output from KRONOS into the MDP must correctly assign probabilities to all states. The probabilities are included in a file which describes the probability of a transition between two states in the original system. Therefore, the parser must identify all instances of this transition in the reachability result and inject the probability accordingly.

For a networked probabilistic timed automaton Kronos generates a composed equivalency graph. This composed graph contains locations which are a union of locations from all individual PTA. To inject probability in the composed equivalency graph, the parser keeps a transition map in the memory and identifies the change of labels between a source state and a destination state. If this change in label is associated with a probability, the net probability of the transition (assigned by default to 1) is updated using multiplication to the new probability. Since equivalency graph contains all the labels from individual transitions being synchronized, this approach ensures that the probability of synchronized transitions is the multiplication of probabilities of individual transitions being synchronized.

For any non-deterministic choices left, the parser flags the transition as non-deterministic so it can be identified and handled correctly by the analysis portion of the program.

5.1. Non-determinism

The conversion of a PTA to a MDP can yield some transitions for which the given probability is not specified. Non-deterministic transitions can be handled in a variety of ways. Our analysis program utilizes three different techniques: minimum, maximum, and average. The minimum technique assumes that every non-deterministic choice will be made by choosing the transition which has the least probability of reaching the reachable set. This technique is useful when the user is attempting to determine a lower bound on the reachability probability. Similarly, the maximum technique chooses the transition to the largest reachability probability so the user can determine an upper bound. The average non-deterministic analysis technique assumes that all outgoing transitions have an equal probability of occurrence, so the transition probabilities of the MDP are adjusted accordingly. This is useful for systems where the non-deterministic transitions truly have equal probability of occurrence. Because this is rare in real systems, the most accurate result is obtained by including as many transition probabilities as possible in the original model.

5.2. Value Iteration

The Value Iteration (VI) technique [20] has been applied to the reachability analysis of a MDP representation of a Stochastic Hybrid System and the convergence results can be found in [16]. This technique is similarly applied for the PTA MDP. Each state of the MDP is assigned a value which represents the reachability probability for that state. Initially this value is one for all states in the goal set and zero for all other states. The VI approach iteratively analyzes each state of the MDP and updates its value by summing the transition probabilities multiplied by the values of the states the transitions lead to for all outgoing transitions of the original state.

The calculation for the value function is described by the equation below where V_i is the value function at state i and $p_{i,j}$ is the probability of state i transitioning to state j. This computation is performed iteratively for every state and repeated multiple times until the values converge.

$$V_i = \sum_j p_{i,j} * V_j$$

$$V_k = 1 \text{ if } k \in GOAL, \text{where } GOAL \text{ is the set of intended reachable states}$$

The values calculated by the VI method at each state of the MDP represent the probabilities for those states to reach the goal set. Therefore, this technique not only answers the reachability question for the initial state, it also answers the question for every other state as well. This occurs because the reachability probabilities for every state are necessary for calculating the final reachability from the initial state, so all probabilities for intermediate states are also available which allows the user more insight into the dynamics of the system.

It can be shown that, the convergent value V_i in each state can be formally described as $V_i = \sum_k P(\pi_k)$, where, π_k is a possible executional trace of the system starting in the ith state and ending in any one of the goal states. Here k is an index set over all possible execution traces leading to the goal state. If $\pi_k = s_0, s_1 \cdots s_n$, then due to markov assumption $P(\pi_k)$ is equal to $\prod_1^{n-1} P(s_{i+1}|s_i)$, where $P(s_{i+1}|s_i)$ is the conditional probability of system moving the state $i+1$ from current ith state, also known as the transition probability.

6. Casestudy -CSMA/CD

In computer networking, Carrier Sense Multiple Access With Collision Detection (CSMA/CD) [1] is a network control protocol typically used for Ethernet. In this protocol, a single universal bus is shared between various senders and receivers by using a mix of carrier sensing scheme, and collision detection scheme. Once the bus detects the transmission of a packet it broadcasts a busy signal that keeps the

[1] http://www.ieee802.org/index.html

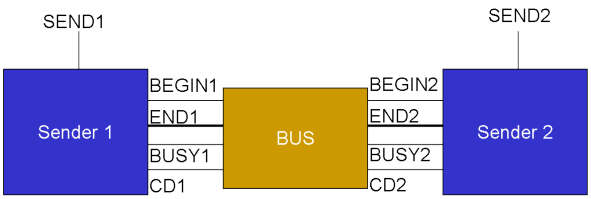

Figure 4. Top Level view of two senders and a bus.

other senders from attempting a transmission. However, since there is a finite delay between the departure of the packet from a sender before it is detected by the bus, there is a chance of a collision. Therefore, if a transmitting sender detects another signal on the bus it stops the transmission and sends out a jam signal, which makes all the senders wait for a random time interval (also known as "back off delay") determined using a truncated binary exponential back off algorithm before trying to send that frame again.

In this case study we formally modeled the CSMA/CD protocol for two scenarios, one with 2 sender processes and the other with 3 sender processes. It is assumed that these sender processes are connected to a single bi-directional $10Mbps$ Ethernet bus. It is known that this bus has a worst case propagation delay σ of $26\mu s$. We only modeled the scenario of fixed length packets and it is known that on the average the total time taken for a successful transmission λ, including the propagation delay is, $808\mu s$. We also assume that the bus does not buffer packets and is virtually error free.

Each sender and the bus are modeled as a probabilistic timed automaton. They communicate via communication channels as shown in Fig. 4. When a collision is detected each sender goes to a collision state and randomly picks one of the possible wait states. Ideally, the sender should reattempt to send the packet more than one time. However, we have only modeled one reattempt by the sender before rejection of the packet to keep the size of the model small.

The basic synchronization events can be summarized as follows:

- $begin_i$: Sender starts transmitting the packet
- $busy_i$: Once the bus detects a packet it communicates busy to all the senders to stop them from attempting transmission.
- end_i: The completion of a transmission from the sender.
- cd_i: If a collision happens then the bus asks the sender to wait for a random time before reattempting transmission.

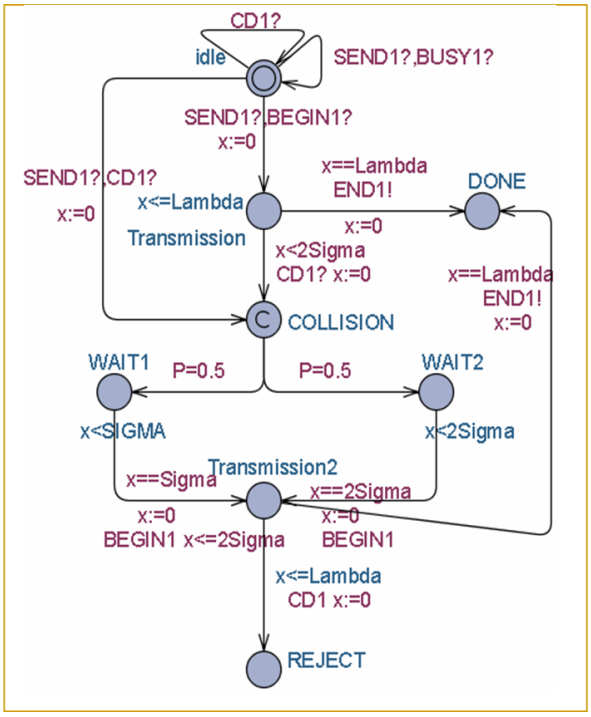

Figure 5. Sender with two wait states.

The probabilistic timed automaton of a sender and a bus are shown in Fig. 5 and 6. The sender is initially idle and goes to the transmission state if a send signal is received from some outside process. During transmission if it receives a cd signal then it moves to a collision state, where it has to immediately pickup one of the possible wait states. The probability of choosing a wait state is 1/number of wait states. This choice is made immediate by setting the invariant of collision state to $x = 0$, where x is the clock associated with the sender. The number of wait states of the sender and the associated probability are changed to model different strategies.

The safety problem was set such that all senders are either in idle state or done state. We did the reachability analysis to find the probability of the success with a 6 wait state strategy and 8 wait state strategy for 2 senders and 3 senders case. It should be noted that the max and min probability for all cases was 1 and 0 respectively.

6.1. Analysis

Table 1 summarizes the analysis result. Figure 7 shows the success result i.e. no collision happening. Note that the results converge as the number of iterations in the value iteration method increase. From the results, it can be seen that probability of success reduces with the number of senders. The result that the success rates increase when the number of wait states are increased for the same number of senders

Table 1. All the computations have been performed on a 4- processor Intel(R) Xeon(TM) CPU 2.80GHz, 1 GB RAM, LINUX Machine. The Propagation Delay is assumed to be $26\mu s$, while the Transmission Delay is assumed to be $808\mu s$

No. of Senders	*No. of Wait States*	*Wait Times (μs)*	*Size of Product Automaton*	*Size of Reachability Graph*	*Avg Probability of Success*	*Time Taken (Real Time/Sys Time)*
2	6	26, 52, 104, 1000, 2000, 3000	94 states, 260 trans, 3 clocks	188 states, 241 trans, 3 clocks	0.8888	$0m0.312s/0m0.000s$
3	6	26, 52, 104, 1000, 2000, 3000	1426 states, 6051 trans, 4 clocks	4602 states, 6471 trans, 4 clocks	0.4819	$3m12.268s/0m0.150s$
2	8	26, 52, 104, 208, 416, 832, 1664, 3328	134 states, 404 trans, 3 clocks	308 states, 391 trans, 3 clocks	0.8020	$0m0.792s/0m0.020s$
3	8	26, 52, 104, 208, 416, 832, 1664, 3328	2310 states, 106747 trans, 4 clocks	8267 states, 11547 trans, 4 clocks	0.4295	$10m5.079s/0m0.250s$

can be attributed to the chosen wait times. In this experiment, wait times were further distributed away from their mean when number of wait states was 6 compared to when the number of wait states was 8.

7. Concluding Remarks

Probabilistic verification of stochastic systems is a useful technique which can provide insights into the intricacies of a real world model. There are many real world systems such as soft real-time systems and biological systems which can naturally have uncertainty and probabilities built into them. Reachability or safety verification can produce results which will help the designer or observer better understand and interact with the system. Our proposed technique of converting the timed automata model to a Markov Decision Process and analyzing the MDP with the Value Iteration method provides a unique and powerful method for probabilistically analyzing these types of systems.

In this paper, we did not compute the bounded time probabilities. However, they can be done by adding an extra clock to the timed automatons to restrict the forward reachability graph.

In the future, this research can follow many directions. This approximation technique has been shown to parallelize elegantly which can significantly reduce the time required to analyze complex systems. There is also promise for developing an analytical solution for special cases of the MDP. Resolving the intricacies dealing with non-determinism could also result in more accurate bounds for the system. Overall, the verification technique proposed has been able to demonstrate its power and flexibility for verifying complex stochastic systems.

8. Acknowledgments

This work was supported in part by DoE SciDAC program under the contract No. DOE DE-FC02-06 ER41442. Sherif Abdelwahed acknowledges support from the NSF SOD Program, contact number CNS-0613971.

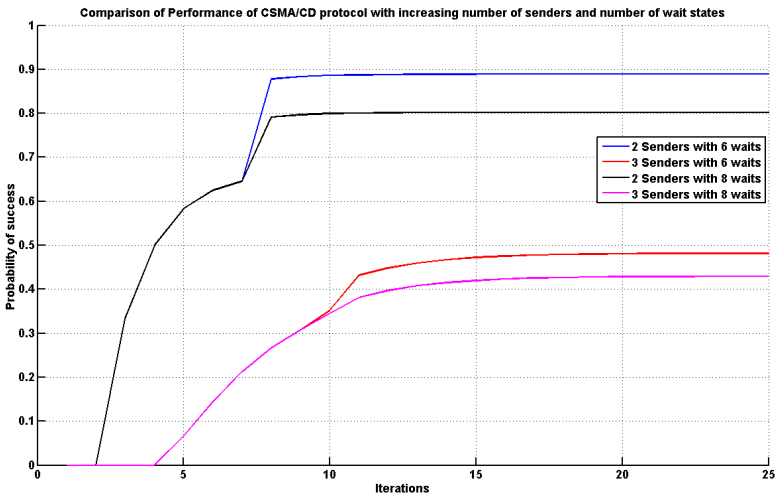

Figure 7. Results.

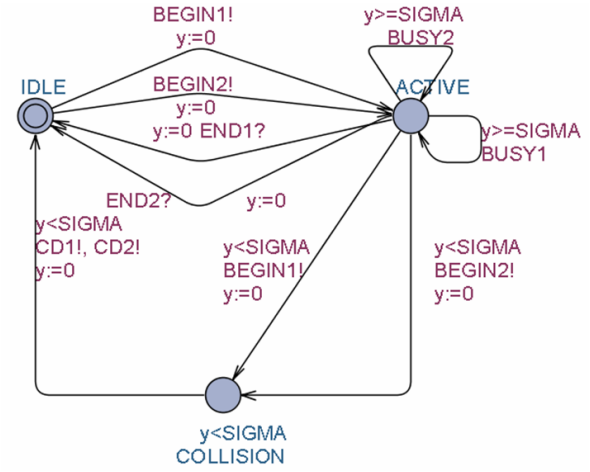

Figure 6. Bus which is used to communicate.

References

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for probabilistic real-time systems (extended abstract). In *Proceedings of the 18th international colloquium on Automata, languages and programming*, pages 115–126, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Z. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *ICALP '97: Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, pages 430–440, London, UK, 1997. Springer-Verlag.

[4] C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distrib. Comput.*, 11(3):125–155, 1998.

[5] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaala tool suite for automatic verification of real-time systems. In *Proceedings of the DIMACS/SYCON workshop on Hybrid systems III : verification and control*, pages 232–243, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[6] A. Bianco and L. de Alfaro. Model checking of probabalistic and nondeterministic systems. In *Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 499–513, London, UK, 1995. Springer-Verlag.

[7] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.

[8] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *CAV*, pages 359–364, 2002.

[9] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. The mobiüs modeling tool. In *PNPM '01: Proceedings of the 9th international Workshop on Petri Nets and Performance Models (PNPM'01)*, page 241, Washington, DC, USA, 2001. IEEE Computer Society.

[10] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge MA, USA, 2000.

[11] L. de Alfaro. Temporal logics for the specification of performance and reliability. In *STACS*, pages 165–176, 1997.

[12] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker. A formal analysis of bluetooth device discovery. *Int. J. Softw. Tools Technol. Transf.*, 8(6):621–632, 2006.

[13] J. W. Haefner. *Modeling Biological Systems: Principles and Applications*. Springer, 2005.

[14] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, January 1995.

[15] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University press, 2 edition, 2000.

[16] X. Koutsoukos and D. Riley. Computational methods for reachability analysis of hybrid systems. In *HSCC '06: Proceedings of the Ninth International Workshop on Hybrid Systems*. Springer-Verlag, 2006.

[17] 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, 2004.

[18] M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. In *ARTS '99: Proceedings of the 5th International AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems*, pages 75–95, London, UK, 1999. Springer-Verlag.

[19] A. Paz. *Introduction to probabilistic automata (Computer science and applied mathematics)*. Academic Press, Inc., Orlando, FL, USA, 1971.

[20] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[21] H. Stark and J. W. Woods. *Probability and random processes with applications to signal processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2002.

[22] M. Steinder and A. S. Sethi. Probabilistic fault diagnosis in communication systems through incremental hypothesis updating. *Comput. Netw.*, 45(4):537–562, 2004.

[23] M. Stoelinga. An introduction to probabilistic automata. *Bulletin of the EATCS*, 78:176–198, 2002.

[24] M. Y. Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *ARTS '99: Proceedings of the 5th International AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems*, pages 265–276, London, UK, 1999. Springer-Verlag.

[25] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 126:110–122, 1997.