Modeling and Verification of Transmission Protocols: A Case Study on CSMA/CD Protocol

Ling Shi
School of Computing
National University of Singapore
Singapore
shiling@comp.nus.edu.sg

Yan Liu
School of Computing
National University of Singapore
Singapore
yanliu@comp.nus.edu.sg

***Abstract*—In this paper, we investigate the modeling and verification of real time systems using a case study on transmission protocol, CSMA/CD. Modeling and verification of real time systems is hot research topic which has practical implications. Such systems are considered as mission critical, as its correctness within timed constraints are of great importance. Through modeling and verifying CSMA/CD protocol using timed extension of CSP (TCSP) modeling techniques in PAT, we propose a formal model for CSMA/CD protocol and verify its critical properties like deadlock-freeness, divergence-freeness and collision detection in a given bounded delay. The integrated reduction techniques in PAT help in carrying out the verification with reasonable speed and results.**

***Keywords*-CSMACD Protocol; Real Time System; Model Checking; PAT**

I. Introduction

Real-time systems are applications to be considered as mission critical. The total correctness of such systems depends not only upon its logical correctness, but also upon the time in which it is performed. Real-time systems can fail in catastrophic ways leading to death or tremendous financial loss. There are many potential causes such as environmental conditions, human errors etc.. However, design errors are increasingly becoming the most serious culprit. Traditional way of verifying software systems has its limitations as human inspection is limited by the abilities of reviewers and only a small portion of the state space of real-time systems can be explored by simulation and testing. Thus, they provide no guarantees about the quality of the system. Formal methods, on the other hand, which refer to mathematically rigorous techniques and tools for the specification, design and verification of software and hardware systems, are often advocated as a way of increasing confidence in such systems [1]. They are able to symbolically examine the entire state space of a system and establish such correctness or safety properties that are true for all inputs. Here, "mathematically rigorous" means that, in a mathematical logic, well-formed statements is used as specifications and rigorous deductions are the basis for formal verifications.

Model checking [2] is a general term for a collection of related formal methods. Such technique offers the potential to guarantee correct functional behavior of a system with respect to its specification. It is studied and adopted in industries as a promising and powerful approach to automatic verification of systems in the last two decades. Our home-grown model checker known as PAT (Process Analysis Toolkit) [3], [4], [5], [6] is such a tool to apply state-of-the-art model checking techniques for system analysis. It supports a wide range of modeling languages including CSP# (short for communicating sequential programs), which shares similar design principle with integrated specification languages like TCOZ [7], [8]. The modeling language integrated in PAT combines high-level modeling operators like (conditional or non-deterministic) choices, interrupt, (alphabetized) parallel composition, interleaving, hiding, asynchronous message passing channel, etc., with programmer-favored low-level constructs like variables, arrays, if-then-else, while, etc.. The real-time system module of PAT is an extension of CSP module with operators which captures quantitative timing requirements. In addition to all above, PAT provides automatic simulation and verification for models, which can help the designer to find implicit errors of a presented model.

Transmission protocols are one kind application of real-time systems, whose rules govern interactions among communication agents. They play an important part in computer networks and distributed systems. Many protocols have been successfully used, but they may suffer from some unexpected failures. The most common faulty in protocols is the occurrence of deadlock; others include loss of message, message destruction, and timeout. In this paper, we formally specify a simple but typical transmission protocol CSMA/CD (Carrier Sense Multiple Access / Collision Detection) [9], which is widely used in Ethernet networks. We have shown how PAT can be used to model this protocol. The protocol properties are also verified based on verification theories in PAT including deadlock freeness, divergence free, timed refinement.

The rest of paper is organized as follows. Section II discusses related works on formal models and verifications of CSMA/CD protocol. Section III introduces timed CSP# language, timed refinement checking for verification and

CSMA/CD protocol, and Section IV specifies CSMA/CD protocol in our modeling language. Section V presents the verification properties and experimental results. Section VI concludes the paper.

II. Related Work

Sergio Yovine conducted case studies on communication protocols in the tool named KRONOS [10]. In the tool's user manual [11], he formally modeled the CSMA/CD protocol using timed automata which captures the system's time constraints in a explicit way, and also he used TCTL to verify important system properties such as reachability, bounded response etc., as well as using timed-abstracting equivalence means to compare a real time implementation of a system with an abstract and untimed specification of it, verifying the correctness of system behaviors. However, using timed automata to model should explicitly set/reset clock variables, this work should be carefully computed, and it's tedious and error-prone. Moreover, timed automata techniques do not support compositional real time systems like deadline, timed-interrupt which is useful in industry case studies.

Another useful model checker Uppaal [12], [13] also demonstrates a model of this protocol which is also based on explicit clock variables. While this model has a deadlock, it does not consider how to respond busy signal to request sender in multi-agents Ethernet networks. In fact, bus just broadcasts busy signal to all senders which causes the deadlock.

Günther Starnberger in his paper [14] using Spin model checker to formally model CSMA/CD protocol. He used five process types to model different behaviors of this protocol and used such processes to make the verification of certain properties straightforward. However, this method requires user to have specialized knowledge and skills in modeling and verifying systems. For example, using processes like monitor process to aid the verification and terminator process to simplify the system model. What's more, the huge memory consumption of visiting the whole state space, like verify the deadlock freeness property needs extra commands to take care of. And lack of ability to check the whole state space make the model checker limited in verifying some important properties when there are many processes. The most inconvenience is that his model does not have time constraints which are very important to protocols in real time systems. Timed properties like time-out, deadline is essential to transmission protocols like CSMA/CD, which would have significant effects on system behaviors.

In previous work [13], the modeling of CSMA/CD protocol is possible to reach a state where there is a deadlock. But such a state does not present in the reality, since with the protocol, the bus network will always be able to receive messages from the senders. Thus in order to model the protocol more precisely, we carefully construct our model in such a way it doesn't present a deadlock state. We model this protocol in the real-time system (RTS) module of PAT, using implicit clocks to model the system behaviors [15]. Implicit clocks have certain benefits that it can model the compositional timed systems, to satisfy high-level system requirements like deadline, timeout, timed interrupt, which can be composed sequentially, or in parallel. We also use timed refinement relationship to check system correctness like KRONOS using timed-abstracting technique. However, our refinement checking is to check whether an implementation satisfies a specification or not. It is different from KRONOS, which uses an extended version of branching time temporal logic named Computation Tree Logic(CTL) [16] with time TCTL [17] to do timed property checking. We also show our verification results of certain critical properties in our home-grown model checker PAT.

III. Background

In this section, we will give the background information of formally modeling and verifying CSMA/CD protocol. It will include knowledge about the modeling language Timed CSP# for modeling real time systems with a comparison to Timed Automata and concept of timed refinement checking for verification of critical properties. At the end of this section, we will briefly introduce the popular transmission protocol- Carrier Sense, Multiple Access with Collision Detection (CSMA/CD) protocol.

A. *Syntax of* Timed CSP#

The Timed CSP# modeling language is a timed extension of Communication Sequence Process (CSP) [18], its grammar is defined as follows.

Definition 1 (Process): A *timed process* is defined by the following grammar.

$$
\begin{array}{lll}
P = \mathit{Stop} \mid \mathit{Skip} & & \text{– primitives} \\
\mid e \rightarrow P & & \text{– event prefixing} \\
\mid [b]P & & \text{– state guard} \\
\mid \mathit{if}\ b\ \mathit{then}\ P\ \mathit{else}\ Q & & \text{– if-then-else} \\
\mid P \,\Box\, Q & & \text{– general choice} \\
\mid P \parallel Q & & \text{– parallel composition} \\
\mid P;\, Q & & \text{– sequential composition} \\
\mid P \setminus X & & \text{– hiding} \\
\mid P \mathrel{\widehat{=}} Q & & \text{– process referencing} \\
\mid \mathit{Wait}[d] & & \text{– delay} \\
\mid P\ \mathit{timeout}[d]\ Q & & \text{– timeout} \\
\mid P\ \mathit{interrupt}[d]\ Q & & \text{– timed interrupt} \\
\mid P\ \mathit{within}[d] & & \text{– react within some time} \\
\mid P\ \mathit{waituntil}[d] & & \text{– wait until} \\
\mid P\ \mathit{deadline}[d] & & \text{– deadline}
\end{array}
$$

where P and Q range over processes, $e \in \Sigma$ is an observable event, b is a boolean expression, X is a set of event names and d is an integer constant.

Stop is the process does nothing but idling, also denotes deadlock. *Skip* states termination. Process $e \rightarrow P$ performs event e first and then behaves as P. Notice that *e* may be an abstract event or a data operation, e.g. written in the form of $e\{x = 1;\ y = 2;\ \}$ or an external C# program. The data operation is used to update data variables and it is assumed to be executed atomically. A guard process is written as $[b]P$. If *b* is true, then it behaves as *P*, else it idles until *b* becomes true. A conditional choice is written as *if b then P else Q*. If b is true, then it behaves P, else it behaves Q. An unconditional choice is written as $P \ \square \ Q$. The choice to choose which process to perform accords to what events are requested by its environment. Parallel composition is written as $P \parallel Q$, where P and Q may communicate via variables, or multi-party event synchronization. Process $P;\ Q$ behaves as P until P terminates and then behaves as Q immediately. $P \setminus X$ hides occurrences of events in X by replacing them with τ (an unobservable event). Process $P \mathrel{\widehat{=}} Q$ defines P to be exactly as Q. Processes may communicate through message passing on channels. Channel buffer size must be greater or equal to 0. Notice that a channel with buffer size 0 sends/receives messages synchronously.

Timed process constructs can be used to capture common real-time system behavior patterns. Process $Wait[d]$ delays the system execution for a period of d time units then it terminates. In process $P\ timeout[d]\ Q$, the first observable event of P should occur before d time units elapse (since the process starts). Otherwise, Q takes control over after exactly d time units elapse. Process $P\ interrupt[d]\ Q$ behaves exactly as P (which may engage in multiple observable events) until d time units elapse, and then Q takes controls over. Process $P\ within[d]$ constrains that P must *react* (by engaging in an observable event) within d time units. Process $P\ waituntil[d]$ denotes P executes for at least d time units and process $P\ deadline[d]$ constrains P must terminate within d time units.

Compared to Timed CSP#, Timed Automata [19] which is popular for specifying real time systems during last decades, has certain deficiencies that it is not feasible in supporting compositional models. Timed Automata are powerful in designing real-time models with explicit clock variables. Real-time constraints are captured by explicitly setting/reseting clock variables. A number of automatic verification supported for Timed Automata have proven to be successful (e.g. UPPAAL [20], KRONOS [10] and RED [21]). However, in industrial case studies of real-time system verification, system requirements are often structured into phases, which are then composed sequentially, in parallel and alternatively [22]. High-level requirements for real-time systems are often stated in terms of deadline, time out, and timed interrupt. Unlike Timed CSP#, Timed Automata lack high-level compositional patterns for hierarchical design. As a result, users often need to manually cast those terms into a set of clock variables with carefully calculated clock constraints. The process is tedious and error-prone.

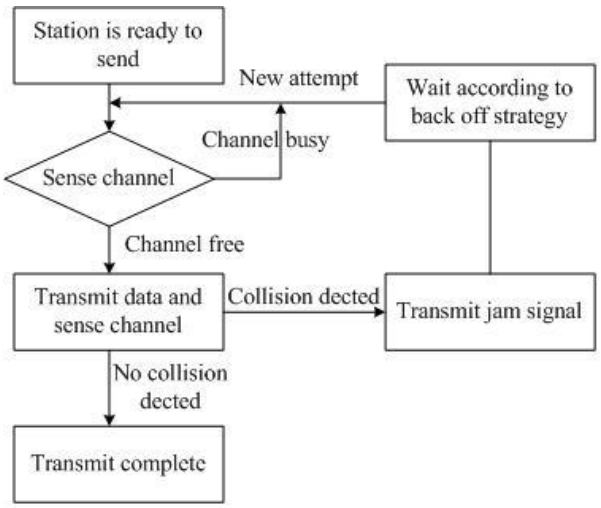

Figure 1. Algorithm of CSMA/CD Protocol

B. Timed Refinement Checking

A timed safety property can be proved by showing a timed trace refinement relationship between an implementation and a handcrafted specification which captures the property. The timed trace refinement relationship [15] is a model satisfies a specification if and only if the timed traces of the models are a subset of those of specification. Assume that a model $\mathcal{I}$ contains two events *start* and *end*. Further, the property is that *end* must occur within 5 seconds since *start* occurs. In order to prove that $\mathcal{I}$ satisfies the property, we can show that $\mathcal{I}$ refines (in timed traces semantics) the following specification: $\mathcal{S} = start \rightarrow ((end \rightarrow \mathcal{S})\ within[5])$.

C. Brief Introduction for CSMA/CD protocol

In Ethernet network, several agents may be connected by a single bus. A problem arises that how to assign the usage of bus to only one of many agents who competes for. Carrier Sense, Multiple Access with Collision Detection (CSMA/CD) protocol describes one solution to this problem. The simplified algorithm of CSMA/CD is shown in Fig. 1. Roughly speaking, whenever an agent starts sending messages, it must first listen to the bus and wait for absence signal before transmitting. When the absence signal comes which means the bus is idle, the agent begins to transmit. If it detects a busy bus, it waits for a random amount of time before another try. As for the propagation time for message to travel from source node to the destination node via bus, an agent may listen to the bus to be idle while another agent is sending message before the message reaches any destination. Thus, collision occurs, then all of the agents are informed of this collision, and abort their transmission immediately. All transmitting messages are lost and all agents should compete for the bus again by waiting a random time.

$$
\begin{aligned}
WaitFor(i) &= (cd?i \rightarrow WaitFor(i)) \\
&\quad \Box\ (newMess!i \rightarrow ((begin!i \rightarrow Trans(i)) \\
&\qquad \Box\ (busy?i \rightarrow Retry(i)) \\
&\qquad \Box\ (cd?i \rightarrow Retry(i)))); \\
Trans(i) &= (cd?i \rightarrow Retry(i) within[0, 52]) \\
&\quad \Box\ (atomic\{end!i \rightarrow Skip\} within[808, 808]; \\
&\quad WaitFor(i)); \\
Retry(i) &= (newMess!i \rightarrow ((begin!i \rightarrow Trans(i) within[0, 52]) \\
&\quad \Box\ (busy?i \rightarrow Retry(i) within[0, 52]) \\
&\quad \Box\ (cd?i \rightarrow Retry(i) within[0, 52])));
\end{aligned}
$$

Figure 2. Model for the *Sender*

Our home-grown model checker PAT integrated the real-time system module which based on the Timed CSP# modeling language makes the modeling and verification of our case study feasible. The parallel verification [23] and on-the-fly refinement checking algorithm [3] enhanced with state space reduction technique enabled the efficient checking of certain crucial properties like safety properties, liveness properties. Thus we explore our modeling and verification of CSMA/CD protocol based on PAT.

IV. Model for CSMA/CD protocol

As in real world, there are several important time parameters, such as different propagation time according to various materials of network wires. In order to better model the real world protocol behavior, we make the following assumptions. First we suppose that agents communicate in the 10Mbps Ethernet with a worst case propagation (denoted here by σ) for absence signal travel of 26 μsec. Additionally, we fix that messages have a fixed length of 1024 bytes, and the time for transmitting a complete message is assumed to be a constant time (denoted here by λ) 808 μsec, including propagation time. Besides, we don't model backoff strategy for retrying, we just assume that agent will retry within 2σ (52 μsec) time unit elapsed since the last step. Also, we make assumptions that no messages are lost during transmitting and there's no buffer for incoming messages at the agent side.

Based on the above assumptions, we then model the CSMA/CD protocol in the real-time system module in our PAT tool. The model for this protocol consists of two components, namely *Sender* (sending agent) and *Bus* (message transmitting channel). *Sender* and *Bus* communicate by synchronous events, so we define this communication by pair-wise synchronization channels. In order to make all the variables and processes of this model to be clearly aware, we list all the related contents of this model with a simplified description, as illustrated in Table I.

Category	Name	Description
Global Definition	N	Constant: number of senders
	channel newMess 0	Sender gets messages to send
	channel begin 0	Sender starts sending message
	channel busy 0	Sender senses a busy bus
	channel cd 0	Sender detects a collision
	channel end 0	Sender completes its transmission
Sender Behavior	WaitFor(i)	Sender i is waiting for a message from the upper level
	Trans(i)	Sender i is sending a message
	Retry(i)	Sender i is waiting to retry after detecting a collision or a busy bus
Bus Behavior	Idle	Bus is free, no sender is transmiting
	Active	One sender starts transmitting and is detecting collision
	Active1	One sender is transmitting messages, bus is busy
	Collision	Collision occurs and bus broadcasts the collision information to all senders

Table I
Components of CSMA/CD Model

A. Modeling Sender Behavior

The behavior of component *Sender* is showed in Fig. 2. *WaitFor* process models the behavior of sender *i* waiting for upper level messages to come. *Trans* process represents sender *i* completes transmitting messages within λ time unit or detects a collision within 2σ (52 μsec) time unit after its sending. *Retry* process expresses sender *i* wait for a 2σ (52 μsec) time unit to re-attempt.

Initially, the sender *i* is in *WaitFor* process. When a message arrives, one of the following transitions is executed. If the bus is not busy, the sender starts transmission. Otherwise, if bus is busy because another sender is already transmitting, it moves to retry state, or a collision is detected, it waits to retry. If a collision occurs while there is no message to send, the sender *i* remains in *WaitFor* state.

In *Trans* process, sender *i* has two transitions, which is modeled as two external choices in PAT. If a collision is detected before 2σ time unit elapsed, the sender goes to *Retry* process. Otherwise, it terminates sending the message after exactly λ time unit, then it goes to the initial process.

When sender *i* is in *Retry* process, it makes a new step to resend messages before 2σ time unit elapsed since the last step. If the bus is idle, it will begin to transmit and moves to *Trans* state; If the bus is busy or receives a collision, it will still be in *Retry* state.

B. Modeling Bus Behavior

The behavior of component Bus is showed in Fig. 3. Initially, bus is in *Idle* process. When one sender starts

$$Idle = newMess?i \rightarrow begin?i \rightarrow Active;$$

$$\begin{aligned} Active = {} & (end?i \rightarrow Idle) \\ & \Box\ (newMess?i \rightarrow \\ & \quad ((begin?i \rightarrow Collision)\ timeout[26] \\ & \quad \Box\ (busy!i \rightarrow Active1))); \end{aligned}$$

$$\begin{aligned} Active1 = {} & (end?i \rightarrow Idle) \\ & \Box\ (newMess?i \rightarrow busy!i \rightarrow Active1); \end{aligned}$$

$$Collision = atomic\{BroadcastCD(0)\}within[0,26];\ Idle;$$

Figure 3. Model for the *Bus*

$$\begin{aligned} BroadcastCD(x) = {} & \textbf{if}\ (x < N)\{ \\ & (cd!x \rightarrow BroadcastCD(x+1)) \\ & \Box \\ & (newMess?[i{==}x]i \rightarrow cd!x \rightarrow BroadcastCD(x+1)) \\ & \} \\ & \textbf{else}\ \{ \\ & \ Skip \\ & \}; \end{aligned}$$

Figure 4. Model for the *BroadcastCD process*

sending its message, bus goes to *Active* process. If bus receives a signal that sender completes sending, it moves to idle state. Or after being in *Active* state for at least σ time unit, bus replies busy signal to any new attempt, which models the fact that the head of the message currently being sent has already propagated, then bus moves to *Active1* state. If another sender starts sending messages before σ time unit elapsed, bus moves to *Collision* state where it takes no more than σ time unit to broadcast collision to all senders. We use atomic process *BroadcastCD* shown in Fig. 4 to broadcast collision to all senders. After that, bus moves to *Idle* state. When bus in *Active1* process, which means a sender has begun sending messages without collision, it will respond busy signal to all request senders until the sender completes transmitting, then bus moves to *Idle* state.

C. Modeling CSMA/CD Protocol

The whole system is executed by all senders and bus interleave with each other. The communication is implemented by the synchronous channel between senders and bus. We model this as Fig. 5

$$CSMACD = (|||\ x : \{0..N-1\}@WaitFor(x))\ |||\ Idle;$$

Figure 5. Model for the *CSMACD protocol*

V. Verification and Experimental Results

A. Verification Properties

In order to formally verify our model for CSMA/CD protocol is correct, we define several categories of properties to check whether it satisfies some properties. These properties in PAT can be categorized as *LTL-X Model Checking*, *Refinement Checking* and *Timed Refinement Checking*. In *LTL-X Model Checking*, properties are formulated using linear temporal logic formulae without next operator, which includes safety property and liveness property. *Refinement Checking* is to verify whether the system satisfies the property by showing a refinement relationship between the system and a model which models the property. The refinement relationship can be trace-refinement, stable failures refinement and failures/divergence refinement [18]. *Timed Refinement Checking* supports refinement checking between timed models, using implicit clocks and zone abstraction mechanism.

Deadlock Freeness (P0)
Informally, safety property states "bad things" never happen during the execution. Deadlock freeness is a safety property that has to be fulfilled so that it is always possible to move from one state to another. Deadlock freeness property in our model is defined as follows:

$$\#assert\ CSMACD\ deadlockfree;$$

Timed Divergence-free (P1)
If a process performs internal transitions and timed transitions forever without engaging any useful events, the process is said to be divergent. While the divergent system is undesirable, for it can give unbound timer, thus disallows timed refinement checking. Timed Divergence-free property in our model is defined as follows:

$$\#assert\ CSMACD\ divergencefree < T >;$$

Collision detection in a given bounded delay (P2)
Whenever two senders are simultaneously transmitting, a collision is detected in a bounded delay. In worst case, a sender can start sending at most σ time units after another sender, which means a collision occurs no more than σ time unit after two senders simultaneously transmit. And collision may take σ time units to be propagated. So a sender will detect a collision at most 2σ (52 μsec) after it starts transmitting.

Figure. 6 shows a model that specifies this property. *Spec* shows that if event $begin.0$ occurs which means sender 0 begins transmitting, then $Constrained1$ happens. $Constrained1$ states if event $begin.1$ occurs thereafter which means sender 1 starts sending messages almost simultaneously, event $cd.0$ or $cd.1$ must occur within 52 time units, otherwise, no constraints apply, which is modeled as *Relaxed* process. In *Spec* process, if event $begin.1$ occurs and then followed by event $begin.0$, then $Constrained2$ happens. $Constrained2$ states if event $begin.0$ occurs thereafter which means sender 0 starts sending messages almost simultaneously, event $cd.0$ or $cd.1$ must occur within 52 time units, otherwise, it executes

$$
\begin{aligned}
&Spec = (newMess.0 \rightarrow begin.0 \rightarrow Constrained1) \\
&\quad \Box\ (newMess.1 \rightarrow begin.1 \rightarrow Constrianed2) \\
&\quad \Box\ Relaxed; \\
&Constrained1 = \\
&\quad ((newMess.1 \rightarrow begin.1 \rightarrow \\
&\quad ((cd.0 \rightarrow Skip \ \Box\ cd.1 \rightarrow Skip)deadline[52]));\ Spec) \\
&\quad \Box\ Relaxed; \\
&Constrained2 = \\
&\quad ((newMess.0 \rightarrow begin.0 \rightarrow \\
&\quad ((cd.0 \rightarrow Skip \ \Box\ cd.1 \rightarrow Skip)deadline[52]));\ Spec) \\
&\quad \Box\ Relaxed; \\
&Relaxed = \\
&\quad (\Box\ x : \{2..N-1\}@(newMess.x \rightarrow begin.x \rightarrow Spec)) \\
&\quad \Box \\
&\quad (\Box\ x : \{0..N-1\}@((newMess.x \rightarrow \\
&\qquad\qquad (busy.x \rightarrow Spec \ \Box\ cd.x \rightarrow Spec)) \\
&\qquad\qquad \Box\ (cd.x \rightarrow Spec) \\
&\qquad\qquad \Box\ (end.x \rightarrow Spec)));
\end{aligned}
$$

Figure 6. Model for the *Collision detection in a given bounded delay*

Relaxed process. In *Spec* process, if no constraints apply, it goes to *Relaxed* process. Our specification is to show whenever two senders send messages simultaneously, they will receive collision within 52 μsec since start transmitting.

In order to verify our model satisfies this property, we use timed refinement to check this requirement. Here, we define this in the following:

$$\#assert\ CSMACD\ refines < T >\ \ Spec;$$

B. Experimental Results

Timed refinement checking allows us to verify Collision detection in a given bounded delay property which consists of timed transitions. We have experimented CSMA/CD protocol on PAT for different number of senders. Table II summarizes the verification results of properties. The experiment testbed is a PC running Windows XP3 within 2.33GHz Intel(R) core(TM)2 Duo CPU and 3.25GB memory.

From the table II, firstly we can see that the number of states, transitions and running time increase rapidly with the number of senders. Secondly, we can show that PAT is effective, for it can handle thousands of states in no more than 1000 seconds. The data on UPPAAL [13] or KRONOS [11] verifying the same models has been omitted from the table because KRONOS just model two senders, and model in UPPAAL [13] has a deadlock, it does not consider how to respond busy signal to request sender in multi-agents Ethernet networks. In fact, bus just broadcasts busy signal to all senders which cause the deadlock. Since our model does not present deadlock state, the more realistic modeling has brought us more states then we can verify our model more correctly.

Property	No. of Senders	Result	#States	#Transitions	Time(sec)
P0	4	Yes	787	1075	0.20
P0	5	Yes	2789	3847	0.60
P0	6	Yes	8851	12227	2.28
P0	7	Yes	26109	35991	8.43
P0	8	Yes	73123	100419	31.03
P0	9	Yes	196997	269319	108.69
P0	10	Yes	514915	700611	361.58
P1	4	Yes	787	1075	0.17
P1	5	Yes	2789	3847	0.66
P1	6	Yes	8851	12227	2.53
P1	7	Yes	26109	35991	9.79
P1	8	Yes	73123	100419	35.69
P1	9	Yes	196997	269319	123.24
P1	10	Yes	514915	700611	407.12
P2	4	Yes	787	1075	0.20
P2	5	Yes	2789	3847	0.90
P2	6	Yes	8851	12227	3.69
P2	7	Yes	26109	35991	14.74
P2	8	Yes	73123	100419	55.38
P2	9	Yes	196997	269319	196.35
P2	10	Yes	514915	700611	655.38

Table II
EXPERIMENTAL RESULTS

VI. CONCLUSION

In this paper, we proposed a formal model for popular transmission protocol named CSMA/CD protocol, which is used in Ethernet to solve the competence for bus recourse among multi-agents. Using our home-grown model checker PAT, we modeled this protocol based on extension of timed CSP and verified its critical properties like deadlock freeness, divergence-freeness, as well as verified this protocol with timed constraints using timed refinement model checking techniques. Experiments show that our model satisfied these properties. We also made comparisons to other model checkers like KRONOS, UPPAAL and Spin to show that PAT is efficient and feasible to model compositional systems. However, our model and verification show there are still certain problems leaving uninvestigated such as the starvation problem should be specially take care of, either by improving our model or making more strict assumptions. Our future work also includes research on other modeling techniques to model richer properties of our systems like applying the probabilistic model checking techniques, and keep on improving PAT to efficiently deal with state explosion problems.

REFERENCES

[1] J. P. Bowen, J. Bowen, J. Bowen, J. Bowen, V. Stavridou, V. Stavridou, V. Stavridou, and V. Stavridou, "Safety-critical systems, formal methods and standards," *Software Engineering Journal*, vol. 8, pp. 189–209, 1993.

[2] O. G. E. M. Clarke and D. A. Peled, *Model Checking*. MIT Press, 2000.

[3] J. Sun, Y. Liu, and J. S. Dong, "Model Checking CSP Revisited: Introducing a Process Analysis Toolkit," in *ISoLA 08*. Springer, 2008, pp. 307–322.

[4] "PAT website," http://www.comp.nus.edu.sg/~pat/.

[5] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "PAT: Towards Flexible Verification under Fairness," in *CAV 2009*, Grenoble, France, June 2009, pp. 702–708.

[6] J. Sun, Y. Liu, J. S. Dong, and H. H. Wang, "Specifying and Verifying Event-based Fairness Enhanced Systems," in *ICFEM 2008*, ser. LNCS, vol. 5256. Springer, 2008, pp. 318–337.

[7] B. Mahony and J. S. Dong, "Timed communicating object z," *IEEE Transactions on Software Engineering*, vol. 26, no. 2, pp. 150–177, 2000.

[8] B. Mahony and J. S. Dong, "Blending object-z and timed csp: An introduction to tcoz," in *THE 20TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE98)*. IEEE Press, 1997, pp. 95–104.

[9] A. S. Tanenbaum, *Computer networks: 2nd edition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[10] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine, "Kronos: A Model-Checking Tool for Real-Time Systems," in *International Conf. on Computer Aided Verification (CAV)*. Springer, 1998.

[11] S. Yovine, "Kronos: A verification tool for real-time systems. (kronos user's manual release 2.2)," *International Journal on Software Tools for Technology Transfer*, vol. 1, pp. 123–133, 1997.

[12] T. Amnell, G. Behrmann, J. Bengtsson, P. R. D;Argenio, A. David, A. Fehnker, T. Hune, B. Jeannet, K. G. Larsen, M. O. Mller, P. Pettersson, C. Weise, and W. Yi, "Uppaal - now, next, and future," in *In Proc. MOVEP00, LNCS 2067*. Springer, 2001, pp. 99–124.

[13] "UPPAAL website," http://www.uppaal.com/.

[14] "Ethernet CSMA/CD verification using the Spin model checker software," https://guenther.starnberger.name/publications/div/spin.pdf.

[15] J. Sun, Y. Liu, J. S. Dong, and X. Zhang, "Verifying Stateful Timed CSP Using Implicit Clocks and Zone Abstraction," in *Proceedings of the 11th IEEEInternational Conference on Formal Engineering Methods (ICFEM 2009)*, ser. Lecture Notes in Computer Science, vol. 5885, 2009, pp. 581–600.

[16] E. Clarke and E. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," 1982, pp. 52–71. [Online]. Available: http://dx.doi.org/10.1007/BFb0025774

[17] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic model checking for real-time systems," *Information and Computation*, vol. 111, pp. 394–406, 1992.

[18] C. Hoare, *Communicating Sequential Processes*, ser. International Series in Computer Science. Prentice-Hall, 1985.

[19] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[20] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," 1997.

[21] F. Wang, R.-S. Wu, and G.-D. Huang, "Verifying timed and linear hybrid rule-systems with red," in *SEKE*, 2005, pp. 448–454.

[22] K. Havelund, A. Skou, K. G. Larsen, and K. Lund, "Formal modeling and analysis of an audio/video protocol: an industrial case study using uppaal," in *RTSS '97: Proceedings of the 18th IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 1997, p. 2.

[23] Y. Liu, J. Sun, and J. S. Dong, "Scalable multi-core model checking fairness enhanced systems," in *ICFEM '09: Proceedings of the 11th International Conference on Formal Engineering Methods*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 426–445.