

Test Generation Framework for Performance Evaluation of Wireless AdHoc MAC Protocols

Shamim Begum, Sandeep Gupta, Ahmed Helmy
Electrical Engineering, University of Southern California
sbegum@usc.edu, sandeep@poisson.usc.edu, helmy@ceng.usc.edu

Abstract

MAC protocol is the main element for determining efficiency in sharing the limited communication bandwidth of a wireless channel. While new protocols and architectures have been designed to address these issues, no systematic approach has been proposed for testing these protocols. Traditional performance evaluation approaches typically evaluate average performance but do not capture the extreme cases. We propose an automatic test scenario generation framework that generates test scenarios for single channel adhoc MAC protocols. As a case study, we use our framework to analyze performance of IEEE 802.11 for adhoc networks. Using our framework we generate library of scenarios in which some nodes in the network suffer from zero throughput while others achieve average throughput resulting in extreme unfairness in IEEE 802.11 networks. Some of our scenarios achieve channel utilization as low as 3%, a 90% reduction compared to utilization obtained for random scenarios that are commonly used for performance analysis. We also perform a comparative analysis of IEEE 802.11 and MACAW based on test results generated by our framework. Empirical analysis of the case studies shows that the complexity of our algorithms is quite manageable for practical purposes.

1 Introduction

Wireless adhoc network is required where a fixed communication infrastructure, wired or wireless, does not exist or has been destroyed. The goal of the network is to allow a group of communicating nodes to setup and maintain connection among themselves without the support of a base station. In recent years, the adhoc network has been very useful in coordination of large scale emergencies, crisis response, military applications, sensor networks, and conference meetings etc. Medium access control (MAC) is the main element to determine how efficiently and fairly the network is utilizing the scarce wireless medium. Sharing a limited bandwidth efficiently and fairly among all nodes are the main objectives of MAC protocols designed for wireless adhoc networks. Issues such as bit error rates, mobility and limited power complicate the design of these protocols. Protocols have been designed to address these issues, however, very few of them have been tested systematically for their performance. There is a pressing need for a systematic approach that exposes network flaws and breaking points. Providing a test generation framework for a broad class of protocols motivates this work.

Test generation (TG) is mainly based on search techniques that search for valid sequences of protocol events that expose weaknesses or errors in the design of a protocol. Traditional test generation approaches target verification and are based on forward search methods where the entire search space is exhaustively searched for test scenarios [1, 2]. Formal verification approaches use high-level system description languages to model and to analyze network protocols [6, 7] and practical systems [5], and involve determining the set of all reachable states of the models. We propose a test generation framework that, instead of adopting the validation approach, uses a “falsification approach” and directly targets the error. Our main formalism is that of a finite state machine (FSM) in which we abstract the atomic representation of protocol events, network node states, and integrate time relations among the events and the state transitions into the abstraction. We define such representation of a system as a **partial scenario**. We start by defining a set of protocol errors as targets for test scenario generation with the goal of exposing protocol weaknesses. An **error** is a partial scenario, and is defined in terms of network node states, protocol events and time relations among the events and the state transitions. We then use a mix of forward and backward search and implication techniques to generate test scenarios that can create

the target error. The implication is used to specify a partial scenario, to identify the components of the scenario that should be precluded from it, and to prune the invalid scenario whenever such components exist in the scenario.

Our proposed error oriented test generation (EOTG) framework is applicable to all single channel wireless adhoc MAC protocols that use handshaking as the basic mechanism to reserve the channel among all the users [11, 12, 13, 14, 15, 16, 17]. The underlying models of various timers, and implication rules in our framework allow it to be used for the broad class of these protocols. We apply our method to the well-known protocols, e.g., IEEE 802.11 [11], MACA [12] and MACAW [13]. As case studies, we present our complete analysis of IEEE 802.11 and a brief analysis of MACAW. Using our framework we have generated libraries of test scenarios that reduce channel utilization to as low as **3%**, and cause some nodes in the network to starve leading to short-term unfairness in IEEE 802.11 network. Short-term fairness is important in many contexts, e.g., smooth acknowledgment flow for TCP and low jitter for real-time audio and video applications [20]. The reduction of channel utilization is more than **90%** compared to utilization used in performance analysis [22]. The scenarios are likely to arise due to a pattern of synchronized transmission events and severely affect the performance of these applications. Based on empirical studies, the performance of our implication algorithms is linear in size of the network for our practical purposes.

The main contributions of this work are twofolds. First, we present an atomic representation of a system using a **partial scenario** that describes protocol events, network node states and their time relations. Such an intuitive representation is simple to abstract a given protocol as well as semantically general for a broad class of protocols. The novelty of our algorithms is in the use of implication rules in specifying the partial scenario and identifying the components to prune an invalid scenario very early in the search process. Second, using our framework we generate scenarios that lead to very low throughput and extreme unfairness in the protocol under study. Such extreme behaviors of a protocol can not be exposed using approaches based on analytical methods, or random simulations. Our systematic test generation approach provides a library of scenarios for simulation that expedite the process of testing various mechanisms throughout the evolution of a protocol. The rest of the paper is organized as follows. Section 2 presents related work. Section 3 describes our proposed EOTG framework. A case study on IEEE 802.11 is presented in Section 4. The case study on MACAW is presented in brief in Section 5. In Section 6 we analyze our framework. Section 7 concludes with a brief outline of our future plans. Detail procedures of our search and implication algorithms are presented in Appendix.

2 Related Work

There is a large body of work on verification and conformance testing. Conformance testing deals with verifying that a specific implementation conforms to a specification, whereas our testing technique is a design (vs. implementation) test. For brevity we refer only the most related and recent literature on the formal verification of network and communication protocols. The main formalism of these works are that of a *probabilistic timed automata* which is an extension of *timed automata* [3]. **PRISM** [4] is a tool for the automatic formal verification of probabilistic systems that uses a high-level system description language to exhaustively analyze the set of all reachable states. It has been used for model checking and for accurate computing of numerical properties of network protocols [6, 7] as well as practical systems [5]. [6] uses PRISM to verify properties of IEEE 802.11 wireless LAN referring both to the likelihood of repeated transmission collision, and to the probability that a node sends packet correctly within a certain deadline. Our work is best positioned with [6] that formally verifies certain properties of IEEE 802.11, whereas, we provide a library of scenarios that exposes some weaknesses of the protocol. Both these approaches are search based framework, and therefore are exhaustive. However, we use implication algorithms to reduce the search space by eliminating invalid branches early in the search process. EOTG framework can be extended for topology synthesis whereas in formal verification approaches, topology must be an input. We discuss topology synthesis in our future work.

Our work is based on STRESS [8] which provides a framework for protocol design and verification. STRESS uses search techniques to synthesize test scenarios given a protocol in a processable form, and its correctness criteria. The result is a set of network topologies, network failures, and protocol event sequences that violate the

correctness criteria. One of the approaches STRESS uses is FOTG (fault oriented test generation) in which the low level fault (e.g., packet loss) is modeled in the form of correctness criteria. Our approach is **error oriented**. An error is a high level (e.g., MAC layer) anomalous behavior and is defined in terms of protocol events, network node states and time relations between protocol events and state transitions. Our approach is closest to FOTG with the following major differences:

- (1) differences in the model: we adopt an atomic representation of protocol messages and states with the semantics of time in order to model behaviors (e.g., collision) at MAC layer that leads to the high level **partial scenario**, whereas, FOTG adopts a non-atomic abstraction with no semantics of time,
- (2) differences in algorithms: we use implication algorithms that are used to deal with prohibited entries in a partial scenario, and
- (3) differences in application: incorporation of semantics of time enriches our TG framework to deal with **partial scenarios** and for the first time allows use of the approach to generate scenarios for wireless network protocols.

[10] presents a survey of 34 MAC protocols for wireless adhoc networks ranging from industry standards to research protocols. Channel access and separation is one of the features on which these protocols are classified. Our framework is designed for single channel MAC protocols where the medium is shared between all nodes in the network for both data and control packets. Handshaking is the basic access mechanism in this class of protocols that includes IEEE 802.11 [11], MACA [12], MACAW [13], FAMA [14], MACA-BI [15], PS-DCC [16], RIMA-SP [17], etc. Our algorithm is applicable for test generation of all these protocols, however, as we present the detailed case study on IEEE 802.11, we only refer to the work related to this protocol.

A number of studies have been conducted to analyze average performance of IEEE 802.11. [18] studies the effect of hidden terminals on performance of IEEE 802.11 using *hearing graph* and *wall model*. The study shows that without RTS/CTS exchange, the throughput, delay and loss for this protocol is very bad which improves significantly when RTS/CTS is used. Our study shows that even with RTS/CTS exchange the protocol performs poorly in some cases. [19] uses a *markov chain* model for performance analysis and shows that the protocol is robust under moderate condition of hidden terminals and node mobility, however, the performance (e.g., throughput) degrades under extreme condition of hidden terminals and node mobility. [21] presents an analysis of IEEE 802.11 with respect to QoS, fairness and performance perspective. The fairness issues we have identified by analyzing the results of our test generation framework are closest to the fairness issues addressed in [21]. We claim that silent drop of RTS packets is one of the main reasons of unfairness in IEEE 802.11 which has been mentioned as one of the three causes leading to unfairness in [21]. To the best of our knowledge, no work has been done that systematically generates these test scenarios for extreme case performance.

3 Proposed EOTG Framework

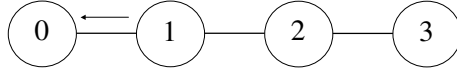
We define an **error** as a set of conditions that describes one or more weaknesses of a given protocol. The error is specified in terms of network node states, protocol events and time relations between the events and the state transitions caused by these events. We start from a **target error** description and search for scenarios that lead to the error. We first identify components of the system under study and model them in a processable representation.

3.1 Base Models

Our overall model consists of (1) a network topology model, (2) models of network node states and protocol messages, (3) a protocol model and (4) an error model.

Network topology: A wireless network is modeled as transmission range of each node in the network. Transmission range of node i is a set G_i where members of the set are nodes who hear its transmission. Figure 1 presents topology I, a wireless network of 4 nodes where $G_0 = \{1\}$, $G_1 = \{0, 2\}$, $G_2 = \{1, 3\}$, and $G_3 = \{2\}$.

Models of network node states and protocol messages: We need to model each protocol message in a way that captures meaningful state transitions the message causes. Transmission of a message m effects the state of transmitter while reception of m effects the states of intended receiver as well as the receivers that overhear



$$G = \{ G_0 = \{1\}, G_1 = \{0, 2\}, G_2 = \{1, 3\}, G_3 = \{2\} \}$$

Figure 1: Topology I: A wireless network of 4 nodes.

it. Therefore, we decompose a protocol message into its transmission and reception. A node transmitting the message m changes its state when it starts the transmission, remains in the state as long as it transmits and changes its state again when it finishes the transmission. Therefore, we further decompose transmission and reception into corresponding *start event* and *end event*. Semantics of an event e from source node i to destination node j is: $e_{i,j}(\gamma)[\sigma]$ where e is the event ID, γ represents its relative timestamp which is the delay of e from the state transition that creates e , and σ represents the set of nodes affected by e . The state s of a node i during period $[t_u, t_v]$ is denoted by $i: \langle s, t_u, t_v \rangle$.

For example, an RTS (request to send) message from node i intended for node j is modeled as following events: $RTS_{i,j}$ -Transmit-Start ($RTS_{i,j}$ -TS) at time t_0 , $RTS_{i,j}$ -Transmit-End ($RTS_{i,j}$ -TE) at $t_2 = t_0 + \alpha_r$, $RTS_{i,j}$ -Receive-Start ($RTS_{i,j}$ -RS) at $t_1 = t_0 + d$ and $RTS_{i,j}$ -Receive-End ($RTS_{i,j}$ -RE) at $t_3 = t_0 + d + \alpha_r$. d denotes message propagation delay and α_r denotes the time it takes to transmit/receive an RTS message. Figure 2 illustrates the semantics of RTS events and corresponding state transitions for topology I. At time t_0 node 1 initiates RTS/CTS exchange with node 0. Times for $RTS_{1,0}$ -TS, $RTS_{1,0}$ -RS, $RTS_{1,0}$ -TE and $RTS_{1,0}$ -RE events are t_0, t_1, t_2 and t_3 respectively. These events are represented as: $RTS_{1,0}$ -TS(0)[{0}], $RTS_{1,0}$ -RS(d)[G_1], $RTS_{1,0}$ -TE(α_r)[{0}] and $RTS_{1,0}$ -RE($d + \alpha_r$)[G_1]. Node 1 changes its state from *Idle* to *Tx* at time t_0 when it starts transmitting the RTS message. At time t_1 , nodes 0 and 2 start receiving RTS message and changes their states from *Idle* to *Rx*. At time t_2 , node 1 finishes its transmission and changes its state from *Tx* to *Wait*. Node 0 finishes receiving the message at time t_3 when it changes its state from *Rx* to *Tx*. Node 2 changes its state from *Rx* to *Defer* at time t_3 . The state history of node 1 is: (1) $\langle \text{Idle}, \text{unknown}, t_0 \rangle$, (2) $\langle \text{Tx}, t_0, t_2 \rangle$ and (3) $\langle \text{Wait}, t_2, \text{unknown} \rangle$. The first entry indicates that node 0 comes out of *Idle* state at time t_0 , however, when it entered this state is unknown at this point. The second entry indicates that it is in *Tx* state from t_0 to t_2 and so on. Similarly the state history of node 0 is: (1) $\langle \text{Idle}, \text{unknown}, t_1 \rangle$, (2) $\langle \text{Rx}, t_1, t_3 \rangle$, and (3) $\langle \text{Tx}, t_3, \text{unknown} \rangle$, and that of node 2 is: (1) $\langle \text{Idle}, \text{unknown}, t_1 \rangle$, (2) $\langle \text{Rx}, t_1, t_3 \rangle$, and (3) $\langle \text{Defer}, t_3, \text{unknown} \rangle$.

The state of a node is one of the following two types: (1) single state, or (2) *combined* state. In single state, a node performs only one function while in *combined* state it performs more than one function. For example, *Rx* is a single state while *WCTSARx* (wait-for-CTS-and-receive) is a combined state in which a node waits and receives at the same time. A protocol event is one of the following two types: (1) non-timer, and (2) timer. A timer event is a an event for which the node has a corresponding timer running in it. We model two types of timers: (1) *non-suppressive* timers, and (2) *suppressive* timers. A suppressive timer is a wait timer which is scheduled to wait for an event and is suppressed when the event occurs.

Protocol Model: We use the above notion of node state and protocol event to model the protocol. The protocol is specified using a transition table F . Each row of transition table defines state transitions of network nodes for a protocol event. Semantics of an entry in the transition table is given by: $\langle s_{in}, e_{j,k}, s_{out}, \{ e1_{j,k}(\gamma_1)[\sigma_1], e2_{j,k}(\gamma_2)[\sigma_2], \dots \} \rangle$. It describes the following transition: event $e_{j,k}$ at any given time t changes the state of node i (where i could be same as j , or k , or it could be different) from s_{in} to s_{out} and triggers events $e1_{j,k}$ at time $t+\gamma_1$, $e2_{j,k}$ at time $t+\gamma_2$ and so on. $e1_{j,k}$ effects a set of nodes σ_1 , $e2_{j,k}$ effects σ_2 and so on.

Error model: We define error E as conditions to expose protocol weakness. E is specified in terms of states of network nodes, protocol events and time relations between state transitions and events. Following is an example of an error for topology I. The error describes a collision at node 1 caused by overlapping of receptions of $RTS_{2,3}$

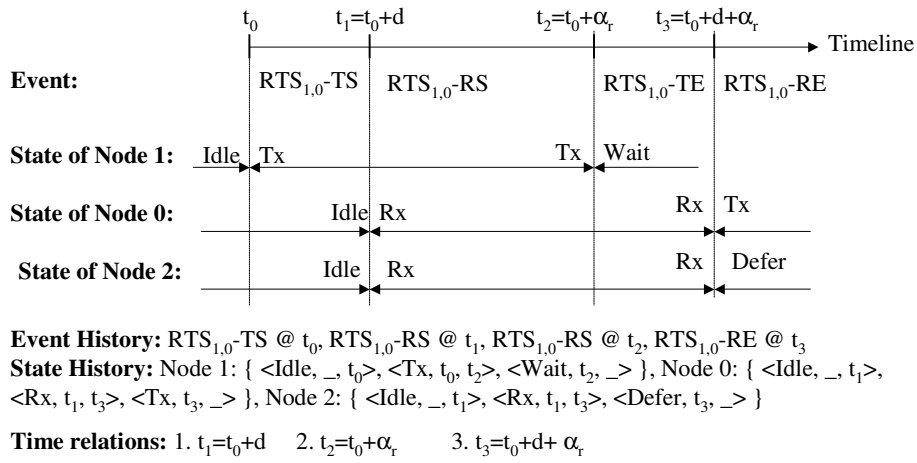


Figure 2: Semantics of a protocol message.

and $RTS_{0,1}$ at the node.

State history: 1: $\langle \text{BOCOL}, t_2, \text{unknown} \rangle$.

Event history: $RTS\text{-RE}_{2,3}$ at t_0 , $RTS\text{-RS}_{2,3}$ at t_1 , $RTS\text{-RS}_{0,1}$ at t_2 .

Time relations:

$$t_1 = t_0 - \alpha_r \quad (1)$$

$$t_1 < t_2 < t_0 \quad (2)$$

In this example, while node 1 receives $RTS_{2,3}$ during time interval $[t_1, t_0]$, it starts receiving $RTS_{0,1}$ at time t_2 at which the node changes its state to *BOCOL*. The time relations/systems of time inequalities (SOI) are solved using mathematical tools, e.g., LINDO. E can be defined for classes of MAC protocols, e.g. collision, unnecessary defer etc.

3.2 EOTG Framework Overview

Figure 3 presents the block diagram of our framework. Inputs to the framework are the transition table F , network topology G and the target error description E . If the error is reachable, our algorithms output all scenarios leading to E . The framework has two building blocks: (1) search and (2) implication. Search algorithms enumerate child nodes and create and maintain the search tree. Implication algorithms derive node state and protocol event history from the states, events and time relations represented by a tree node and check consistency of the tree node. Inconsistent tree nodes are pruned.

We describe our search and implication algorithms using a protocol P . Table 1 presents the description of states and events of the protocol. Note that *Idle* is an initial state and *Pkt* is an initial event. Among the states, only *WCTSARx* is a combined state. Among the timers, *WCTS* timer is a suppressive timer as it gets suppressed by reception of the corresponding *CTS*. Table 2 and 3 presents the time variables and the transition table of the protocol P , respectively. The protocol works as follows: upon receiving a packet from higher layer for node j , node i transmits an *RTS* destined for node j , schedules a *WCTS* timer to wait for the *CTS* from j . Upon receiving the *RTS* destined for it, node j transmits a *CTS* to node i . While overhearing the *RTS*, a node k defers access to the channel by scheduling a timer for a period of θ_r . If node i receives the *CTS* from j before its *WCTS* timer expires, it goes to *Idle* state, otherwise, it retransmits the *RTS*. Note that the type of a timer can be determined from the transition table.

3.3 Search algorithms and concepts

We refer to network node state or protocol event using a common term *entity*.

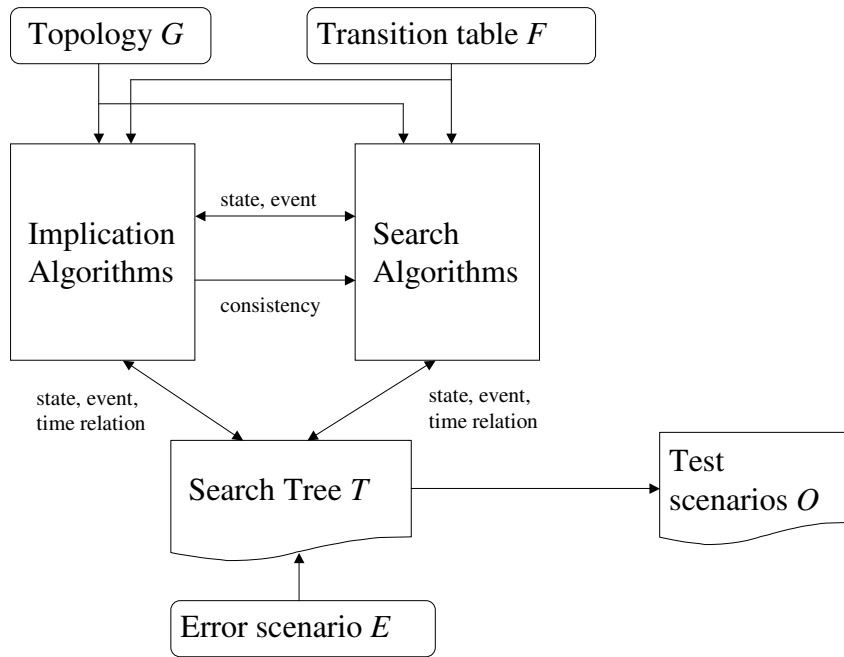


Figure 3: Block diagram of EOTG framework.

Scenario: A node T_n in search tree T describes a scenario. It is defined by the following: history or description of network node states (denoted by H_s), history of protocol events (H_e), time relations or system of time inequalities (SOI), and prohibited lists. The first three describe the scenario presented by T_n while the prohibited lists describe network node states (PL_s), protocol events (PL_e) and time relations (PL_{SOI}) between prohibited and history entries that are prohibited or precluded from the scenario. Figure 5.(a) presents a collision scenario of protocol P in topology I . The scenario describes collision of $RTS_{2,3}$ and $RTS_{0,1}$ at node 1. While node 1 is receiving (overhearing) $RTS_{2,3}$ during interval $[t_1, t_0]$, it starts receiving $RTS_{0,1}$ at time t_2 leading to a collision at t_2 . Figure 5.(a) is an input error description of a collision and does not have any entries in the prohibited list. The scenario in Figure 5.(b) consists of two event entries in PL_e which are precluded from the scenario as long as they satisfy the time relation presented in PL_{SOI} . In this paper, we use *tree node* T_n and the *scenario* described in the node synonymously.

Compatible entities: Two events are said to be *incompatible* if their event IDs are different, or sources are known and different, or destinations are known and different. Otherwise, the events are *compatible*. For example, $RTS-RS_{1,0}$ and $RTS-RS_{1,unknown}$ are compatible, but $RTS-RS_{1,0}$ and $RTS-RS_{1,2}$ are incompatible. Two states are *compatible* if their node IDs are same and state IDs are same. Otherwise, they are *incompatible*. Given two compatible entities, x at time t_x and y at t_y , where y exists in scenario T_n , the existence of x in T_n (with respect to y) can be determined using time stamp test **test-exist** presented in Table 4.

Lemma 1. *Test test-exist accurately determines existence of entity x at t_x in a given scenario T_n .*

Proof. Assume x exists in T_n and the only compatible entity is y . Therefore, t_x must be the same as t_y leading to a solution to the $t_x=t_y$ test, no solution to $t_x<t_y$ and $t_x>t_y$ tests. If x does not exist in T_n , there will be no compatible entity y such that $t_x=t_y$ test has a solution. In all other cases, it is undecidable. \square

Predecessor of entity: If entity y is sufficient to create an entity x , entity y is said to be a predecessor of x . Note that an entity may have one or more predecessors except the initial state and the initial event. Procedure

Table 1: States and events of protocol P.

State name	State description	Type
Idle	Does nothing	Initial, single
Tx	Transmitting	Single
Rx	Receiving	Single
WCTS	Waiting for CTS	Single
Defer	Deferring access to channel	Single
WCTSARx	Waiting for CTS and receiving	Combined
BOCOL	Backoff on collision detection	Single
Event name	Event description	Type
Pkt	Packet from higher layer	Initial, non-timer
RTS-TS	Request-To-Transmit Transmission Start	Non-timer
RTS-RS	Request-To-Transmit Reception Start	Non-timer
RTS-TE	Request-To-Transmit Transmission End	Non-timer
RTS-RE	Request-To-Transmit Reception End	Non-timer
CTS-TS	Clear-To-Transmit Transmission Start	Non-timer
CTS-RS	Clear-To-Transmit Reception Start	Non-timer
CTS-TE	Clear-To-Transmit Transmission End	Non-timer
CTS-RE	Clear-To-Transmit Reception End	Non-timer
WCTST-TS	Wait for CTS Timer Start	Suppressive timer
WCTST-TE	Wait for CTS Timer End	Suppressive timer
DeferT-TS	Defer Timer Start	Non-suppressive timer
DeferT-TE	Defer Timer End	Non-suppressive timer
BOCOLT-TS	Backoff Timer Start	Non-suppressive timer
BOCOLT-TE	Backoff Timer End	Non-suppressive timer

Table 2: Time variables of protocol P.

Time variables	Description
α_r	Time to transmit/receive RTS
α_c	Time to transmit/receive CTS
d	Propagation delay
θ_r	Defer period on RTS
w_c	WCTS period
B_c	Backoff period

create-pred presented in Appendix creates the predecessor set P_x of a given entity x .

Justification of entities: An entity x in tree node T_n is **justified** if at least one of its predecessors exists in the scenario T_n . The entity is **unjustified** if none of its predecessors exists in the scenario. If all entities of a scenario are justified, it is called a **fully specified scenario**. If one or more entities of the scenario are unjustified, it is called a **partially specified scenario** or **partial scenario**.

Enumeration procedure: Procedure **check-existence** checks the existence of an entity in a scenario. Given x , we first use procedure **create-pred** to create P_x , predecessor set of x and then check existence of each of these predecessors using procedure **check-exist**. If at least one of the predecessors in P_x is confirmed to exist in the scenario, x is determined to be justified. In other words, if none of the predecessors of the set P_x exists in the scenario, x is determined to be **unjustified**. Given a scenario T_n , we consider the unjustified entities to create its child nodes. We take cross product between predecessor sets of all unjustified entities to create child nodes of T_n . Procedure **enumerate** enumerates all child nodes for a tree node T_n .

Lemma 2. *Given a tree node T_n , procedure **enumerate** enumerates all child nodes rooted at T_n .*

Proof. Given an entity x in T_n , procedure **create-pred** creates all its predecessors from the transition table. It directly follows from Lemma 1 that we can determine whether x is justified in T_n accurately. Procedure

Table 3: Transition table of protocol P.

	Start state	Input event	End state	Output event
1	Idle	Pkt _{<i>i,j</i>}	Tx	RTS-TS _{<i>i,j</i>} (0)[{ <i>i</i> }], RTS-RS _{<i>i,j</i>} (<i>d</i>)[G _{<i>i</i>}], RTS-TE _{<i>i,j</i>} (α_r)[{ <i>i</i> }], RTS-RE _{<i>i,j</i>} (<i>d</i> + α_r)[G _{<i>i</i>}]
2	Tx	RTS-TE _{<i>i,j</i>}	WCTS	WCTST-TS _{<i>i,i</i>} (0)[{ <i>i</i> }], WCTST-TE _{<i>i,i</i>} (<i>w_c</i>)[{ <i>i</i> }]
3	Idle	RTS-RS _{<i>j,i</i>}	Rx	
4	Rx	RTS-RE _{<i>j,i</i>}	Tx	CTS-TS _{<i>i,j</i>} (0)[{ <i>i</i> }], CTS-RS _{<i>i,j</i>} (<i>d</i>)[G _{<i>i</i>}], CTS-TE _{<i>i,j</i>} (α_c)[{ <i>i</i> }], CTS-RE _{<i>i,j</i>} (<i>d</i> + α_c)[G _{<i>i</i>}]
5	WCTS	CTS-RS _{<i>j,i</i>}	WCTSARx	
6	WCTSARx	CTS-RE _{<i>j,i</i>}	Idle	
7	Idle	RTS-RS _{<i>j,k</i>}	Rx	
8	Rx	RTS-RE _{<i>j,k</i>}	Defer	DeferT-TS _{<i>i,i</i>} (0)[{ <i>i</i> }], DeferT-TE _{<i>i,i</i>} (θ_r)[{ <i>i</i> }]
9	WCTS	WCTST-TE _{<i>i,i</i>}	Tx	RTS-TS _{<i>i,j</i>} (0)[{ <i>i</i> }], RTS-RS _{<i>i,j</i>} (<i>d</i>)[G _{<i>i</i>}], RTS-TE _{<i>i,j</i>} (α_r)[{ <i>i</i> }], RTS-RE _{<i>i,j</i>} (<i>d</i> + α_r)[G _{<i>i</i>}]
10	Rx	RTS-RS _{<i>j,i</i>}	BOCOL	BOCOLT-TS _{<i>i,i</i>} (0)[{ <i>i</i> }], BOCOLT-TE _{<i>i,i</i>} (<i>B_c</i>)[{ <i>i</i> }]

Table 4: **test-exist** to determine existence of x w.r.t. y .

EQ Test ($t_x = t_y$)	SLT Test ($t_x < t_y$)	SGT Test ($t_x > t_y$)	test-exist output	Remarks
Unsolvable	don't care	don't care	x New	
Solvable	Unsolvable	Unsolvable	x Old	
Solvable	Unsolvable	Solvable	x May-be-old	If old, $t_x = t_y$
Solvable	Solvable	Unsolvable	x May-be-old	If old, $t_x = t_y$
Solvable	Solvable	Solvable	x May-be-old	If old, $t_x = t_y$

enumerate only considers unjustified entities. Cross product of the predecessor sets of all unjustified entities enumerates all possible combinations of predecessors, and there all child nodes rooted at T_n . \square

Test scenario generation procedure: Given an error scenario described in E , we generate test scenarios leading to E using procedure **create-tree**. We apply a depth-first search (DFS) technique and use procedure **enumerate** to enumerate child nodes. The search continues until (1) we reach at a *leaf node* at which all entities are justified, or (2) the scenario is pruned using implication rules presented in Section 3.4.

Lemma 3. *All entities are justified in a leaf node. In other words, a leaf node of a search tree is indeed a fully specified scenario.*

Proof. Let entity x is an unjustified entity in a leaf node T_n . From the definition of justification, no predecessor of x exists in T_n . Now, procedure **enumerate** must create predecessors of x to enumerate child nodes rooted at T_n that contradicts with the initial assumption of T_n being a leaf node. \square

Lemma 4. *Given an error scenario described in E , procedure **create-tree** is guaranteed to generate a test scenario if it leads to E . It is guaranteed to prune a scenario if it does not lead to E .*

Proof. See Section 6.1 of this paper. \square

3.4 Implication rules and algorithms

Implication is the process of determining network node state and event history as a consequence of a given scenario. If an entity x in scenario T_n can uniquely ¹ be created from entity y , then **backward implication** of x results y . Entity y is added to the scenario as a result of backward implication. If x uniquely creates entity z , i.e., z can uniquely be created from x , then **forward implication** of x adds z to T_n . We use implications to specify a partially specified scenario. While specifying a scenario, our implication rules generate **prohibited entities** that are precluded from the scenario. For example, consider the input error description presented in Figure 5.(a) that describes a collision of RTS_{2,3} and RTS_{0,1} at node 1. In Figure 5.(b), implication of event RTS-RE_{2,3} generates

¹the only way in which entity x can be created in T_n is by entity y

prohibited entries in PL_e and PL_{SOI} using implication rules we describe in this section. These entries in PL_e are precluded from the scenario as long as they satisfy the time relation presented in PL_{SOI} . In the process of implication, whenever an entity generated to be added to the history (H_s or H_e) is determined to exist in the **prohibited list** (PL_s or PL_e), or vice versa, we prune the scenario. We also prune using a state consistency check presented later in this section. Thus the function of implication is to specify a partially specified scenario as well as to prune inconsistent branches of the search tree.

Following is a complete list of implication rules used in our algorithms. Rules 3, 5, 6, and 9 are used to generate prohibited entities, and to check their existence in the history in order to eliminate invalid scenarios. Therefore, correctness of these rules ensures the correctness as well as the completeness of our algorithms. For this reason, we present these rules using Lemma 5, 6, 7, and 8, respectively. In Appendix, we present the procedures to generate the prohibited entities, to check their existence, and to eliminate invalid scenarios according to these rules.

1. Transmission-Reception rule: Reception of a message m implies the transmission of m . A reception-start (RS) event implies (backward) a transmission-start (TS) event. Assuming no loss, a TS event implies (forward) corresponding RS event.

2. Event start-end rule of a message: An end event implies (backward) the start event for the corresponding message. If we encounter a transmission-end (TE) event, it is implied that a transmission-start (TS) event was encountered earlier. Similarly, an RE event implies corresponding RS event. Assuming the wireless nodes do not fail, a TS event implies (forward) a TE event. Note that an RS event does not *always* imply (forward) corresponding RE event because a receiver receiving the message may receive other message causing a collision and hence a garbled reception.

3. Successful reception rule: A reception of a message in a scenario T_n is confirmed as *successful* when the corresponding RE event is generated in T_n . Note that according to rule 2, forward implication of an RS event does not generate the corresponding RE event. Therefore, an RE event is generated either by backward implication or by **create-pred** procedure, and always is implied as *successful*. We present the implication of a successful reception using Lemma 5.

Lemma 5. *Given a successful reception of a message m defined by $m-RE_{i,j}$ at time t_v at node k and corresponding $m-RS_{i,j}$ at time t_u in a tree node T_n , there should not be any event $n-RS_{p,q}$ at time t_w or any event $n-RE_{p,q}$ at time t_w in T_n such that n is a message of the protocol, k is an element of the sets G_i and G_p , and $t_u \leq t_w \leq t_v$.*

Proof. Let there exists an event $n-RS_{p,q}$ at time t_w in the scenario T_n during the successful reception interval $[t_u, t_v]$. From the definition of collision, at time t_w the receiving node k must change its state to the collision state *BOCOL* leading to a garbled reception that contradicts our initial assumption of successful reception. \square

A successful reception of message m from node i to j at node k for a reception interval of $[t_u, t_v]$ implies that there was no other reception in the neighborhood that node k hears during the interval. If any such event does exist in the scenario, we prune the branch. Procedure **succ-rx** generates these prohibited entities in PL_s , PL_e and PL_{SOI} . It also checks existence of these prohibited entities in the scenario T_n . If exists, it returns to prune the branch.

4. Timer expiration rule: A timer expiration event implies a timer set event. If a timer expires at time t , it is implied that it was set at time $t-\delta$ where δ is the period for which the timer was scheduled. While, on the other hand, if a timer is set at time t , it does not imply that it would expire at time $t+\delta$. The forward implication depends on the type of timer. For non-suppressive timers, forward implication holds as they expire when the scheduled period (δ) is over. For suppressive timers it does not hold. Suppressive timers may get suppressed before $t+\delta$.

5. Suppressive timer rule: A suppressive timer is scheduled to wait for some event to occur and is get suppressed when the event occurs. These timers are wait timers. We present the implication of a suppressive timer expiration using Lemma 6.

Lemma 6. *Given a suppressive timer mT with the timer set event $mT-TS_{i,i}$ at time t_u and the timer end event $mT-TE_{i,i}$ at time t_v at node i in a tree node T_n , there should not be any event $e_{j,k}$ at time t_w in T_n such that n is the event of the protocol for which the timer mT is waiting, j and k are two nodes of the network (and one of them could be the same as i), and $t_u \leq t_w \leq t_v$.*

Proof. Let there exists an event $e_{j,k}$ at time t_w in T_n during interval $[t_u, t_v]$ for which the timer was waiting. From the definition of suppressive timer, the timer must time out at time t_w ($< t_v$) that contradicts the scheduled timer expiration event at time t_v . \square

Expiration of timer mT at time t_v which was set at time t_u for an event $e_{j,k}$ to occur during interval $[t_u, t_v]$ implies that the event $e_{j,k}$ did not occur during the interval. If any such event does exist in the scenario, we prune the branch. Procedure **supp-timer** generates these prohibited entities in PL_s , PL_e and PL_{SOI} . It also checks existence of these prohibited entities in the scenario T_n . If exists, it returns to prune the branch.

6. Non-suppressive timer rule: Generally in MAC protocols, non-suppressive timers are designed to control the access to the channel. For example, NAV (we refer it as defer timer) and backoff timers are designed to defer access to the channel. We present the implication of a non-suppressive timer using Lemma 7.

Lemma 7. *Given a non-suppressive timer mT defined by timer expiration event $mT-TE_{i,i}$ at time t_v and set event $mT-TS_{i,i}$ at time t_u in a tree node T_n , there should not be any event $n-TS_{i,j}$ at time t_w or any event $n-TE_{i,j}$ at time t_w in T_n such that n is a message of the protocol, j is an element of the set G_i , and $t_u \leq t_w \leq t_v$.*

Proof. Let there exists an event $n-TS_{i,j}$ at time t_w during the interval $[t_u, t_v]$ in T_n . From the definition of non-suppressive timer, the event must have occurred either before the interval or after the interval that contradicts our initial assumption of the event $n-TS_{i,j}$. \square

A non-suppressive timer mT at node i for a interval of $[t_u, t_v]$ implies that there was no transmission from node i to any of its neighbor during the interval. If any such event does exist in the scenario, we prune the branch. Procedure **non-supp-timer** generates these prohibited entities in PL_s , PL_e and PL_{SOI} . It also checks existence of these prohibited entities in the scenario T_n . If exists, it returns to prune the branch.

7. State creation rule: End state.output events \leftarrow Start state.input event. This rule states that if node i is in **start state** when it encounters the **input event**, it changes its state to **end state** and triggers the **output events**. This rule is used to create state history from states and events.

8. Rule of neighborhood: When a reception event is encountered by a node j where the transmission is from node i such that j is an element of the set G_i , the same reception event must be encountered by other nodes of the set G_i unless a loss occurs.

9. Rule of state consistency: The state representation in our model is such that a node remains in only one state (single or combined state) during an interval of time. Therefore, existence of two different states of a node during any interval leads to a state inconsistency. We present this result as Lemma 8.

Lemma 8. *If a node i is in state s_1 from time t_1 to t_2 and in state s_2 from time t_3 to t_4 such that $s_1 \neq s_2$, then it leads to inconsistency if these time intervals overlap.*

Proof. If the intervals overlap, the node remains in both states s_1 and s_2 during the period the intervals overlap and therefore contradicts our basic combined state representation. \square

Table 5 presents **test-interval-overlap** test to determine if two given intervals overlap. Procedure **state-cons** checks the consistency of state history in a given scenario. The intervals $[t_1, t_2]$ and $[t_3, t_4]$ are guaranteed to overlap if **both** of the non-overlapping tests NOT1 ($t_2 < t_3$) and NOT2 ($t_2 < t_3$) do not have any solution.

Implication procedures: We first present the basic idea of implication using an example of protocol P described in Table 3. Let us perform backward implication of the event RTS-TS_{0,1} at t_0 in the scenario presented in Figure 4. Event RTS-TS is output in row 1 and row 9 in transition table presented in Table 3. Hence we

Table 5: **test-interval-overlap** to test whether intervals $[t_1, t_2]$ and $[t_3, t_4]$ overlap.

NOT1 Test ($t_2 < t_3$)	NOT2 Test ($t_4 < t_1$)	Result
Solvable	Don't care	May not overlap
Don't care	Solvable	May not overlap
Unsolvable	Unsolvable	Must overlap

H_s: 0: <Idle, t ₁ , t ₂ >	SOI: t ₁ < t ₂ t ₂ ≤ t ₀	PL_e: 0: <WCTS, t ₃ , t ₄ >	PL_{soi}: t ₃ < t ₂ t ₄ ≤ t ₀
RTS-TS _{0,1} @ t ₀			

Figure 4: An example scenario of protocol P .

have two predecessors, p_1 and p_2 of the event. In order for the predecessor p_1 to be the one that triggers this event, node θ must (1) be in state *Idle* for a period $[t_a, t_b]$ such that $t_a < t_0$ and $t_b \geq t_0$ and (2) not be in state *WCTS* for a period $[t_u, t_v]$ such that $t_u < t_0$ and $t_v \geq t_0$. Condition 1 confirms the necessary state in which the input event *Pkt* can trigger the state transition to generate the RTS-TS event while condition 2 rules out the predecessor p_2 . Lemma 9 formally states this process of unique implication.

Lemma 9. *Given an entity x that has N predecessors, it is guaranteed that the predecessor p_k uniquely creates x if only p_k satisfies the condition of existence, and all other $N-1$ predecessors either satisfy the condition of non-existence or are ruled out.*

Proof. Condition of existence for only predecessor p_k is a necessary condition for unique implication. The condition of non-existence for all other entities rules out other predecessors to be derived by implication, and therefore is sufficient in order for p_k to be the unique implication of x . \square

In the above example, backward implication of RTS-TS_{0,1} generates event Pkt_{0,1} at t_0 and state 0: < $Tx, t_0, unknown$ > because of the unique implication. If more than one predecessors satisfy the condition of existence, then we prune the branch. If none of the predecessors satisfies the condition, we continue without doing any implication at this point. In doing implication of an entity, we first do the process of elimination described above (procedure **elim-rows** to eliminate predecessors. If number of predecessors left after elimination is more than one, we check if there exists an entity common to all the predecessors that we can derive uniquely (procedure **row-intersection**). Procedure **bk-implication** presented in Appendix performs the backward implication of a given entity.

Scenario specification using implications:

Given a scenario presented in a tree node T_n , procedure **specify-scenario** applies both backward and forward implications on all entities of the scenario as long as the implication procedures generate new information. In other words, it stops performing implication at a step (or round) k if no new information is generated in the k^{th} implication step. Note that procedure **fw-implication** is similar to procedure **bk-implication** with the exceptions that it checks in the transition table where the entity is input.

3.5 An Example

Given a scenario, we first do implication of events and states as long as the implication procedures add new information into the scenario. Implication of an entity may add entity (event, state, time relation) into the scenario

Table 6: **find-relation** to determine relationship of time stamps.

EQ Test ($t_u = t_v$)	SLT Test ($t_u < t_v$)	SGT Test ($t_u > t_v$)	Output
Solvable	Unsolvable	Unsolvable	$t_u = t_v$
Solvable	Solvable	Unsolvable	$t_u \leq t_v$
Solvable	Unsolvable	Solvable	$t_u \geq t_v$
Solvable	Solvable	Solvable	Unknown
Unsolvable	Unsolvable	Unsolvable	Unknown
Unsolvable	Solvable	Unsolvable	$t_u < t_v$
Unsolvable	Unsolvable	Solvable	$t_u > t_v$
Unsolvable	Solvable	Solvable	Unknown

if it does not exist in the scenario. The scenario is pruned if the entity exist in the corresponding prohibited list with certainty (i.e., if the **check-existence** procedure for that entity returns a “Yes” to check against the prohibited list). Consider an error description E that describes a collision of $RTS_{2,3}$ and $RTS_{0,1}$ at node 1 of topology I presented in Figure 5.(a). We start by copying the error description E into T_0 , the root of the search tree.

Implication of the events in the scenario adds the corresponding transmit-start (TS) events into H_e , timer relations into SOI , and prohibited entries into PL_e and PL_{SOI} as presented in Figure 5.(b). In the figure, we only show the entities that are added to the scenario (i.e., the entities that are generated by implication, but are not added to the scenario are not shown). For example, event entry $RTS-RE_{2,3}$ at t_0 is a successful reception at node 3 and by Lemma 5, there should be no other reception event that node 3 can potentially hear (or overhear) during the successful reception interval $[t_1, t_0]$. Since node 3 has only one neighbor (node 2), the two events are added in the prohibited list in Figure 5.(b). Whenever an entry is added to the prohibited list, our procedure checks if the entry exist in the history (H_e and H_s). If the prohibited entry exists in the history, the scenario is pruned. In this example, the prohibited entries do not exist in the history. Now, events $RTS-RS_{2,3}$ at t_1 and $RTS-RS_{0,1}$ at t_2 generate $RTS-TS_{2,3}$ at t_3 and $RTS-TS_{0,1}$ at t_4 respectively using implication rule 1. Figure 5.(c) and 5.(e) present the scenario after the first and second round of implications, respectively.

Figure 6 presents the scenario after it is specified using 3 rounds of implication at which point no new information is generated by the implication. Therefore, we enumerate the tree node T_0 at this point. Note that the enumeration procedure returns all child nodes of T_n , however, we continue with one branch at a time to perform a DFS search. Table 7 presents the justification status of event history of the scenario presented in Figure 6. The column denoted by **Total predecessor** presents the number of predecessors of the event as derived from the transition table in Table 3. The column denoted by **check-existence** presents the existences status of the predecessor of the event as returned by the **check-existence** procedure. For example, $RTS-RE_{2,3}$ at t_0 has only 1 predecessor that exists in the scenario ($RTS-RE_{2,3}$ at t_1), and hence it is justified. Events $RTS-TS_{2,3}$ at t_3 and $RTS-RE_{0,1}$ at t_4 are unjustified (**bold** in Figure 6) as none of their predecessors exists in the scenario.

Table 7: **Justification status of event history of scenario in Figure 6.**

Event	Total predecessor	check-existence	Justification status
$RTS-RE_{2,3}$ at t_0	1	Yes	Justified
$RTS-RS_{2,3}$ at t_1	1	Yes	Justified
$RTS-RS_{0,1}$ at t_2	1	Yes	Justified
$RTS-TS_{2,3}$ at t_3	2	No	Unjustified
$RTS-TS_{0,1}$ at t_4	2	No	Unjustified
$RTS-TE_{2,3}$ at t_5	1	Yes	Justified
$RTS-TE_{0,1}$ at t_6	1	Yes	Justified
$WCTST-TS_2$ at t_5	1	Yes	Justified
$WCTST-TE_2$ at t_6	1	Yes	Justified

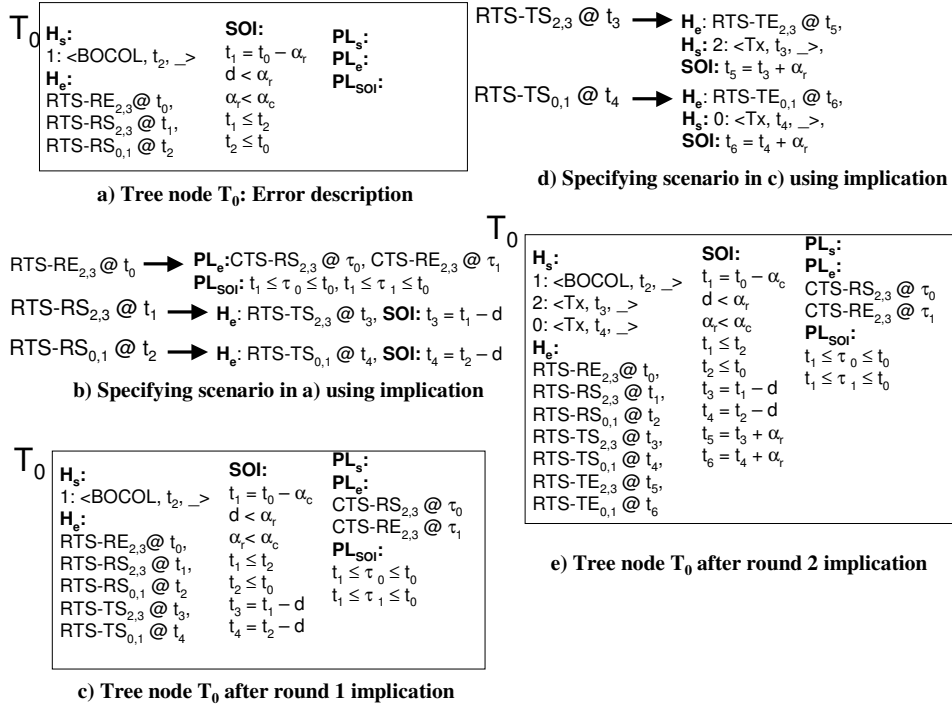


Figure 5: A collision description of protocol P .

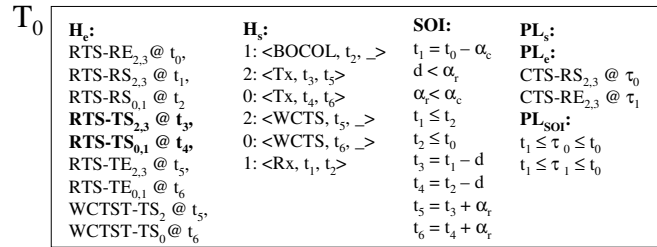


Figure 6: The scenario after 3 rounds of implications.

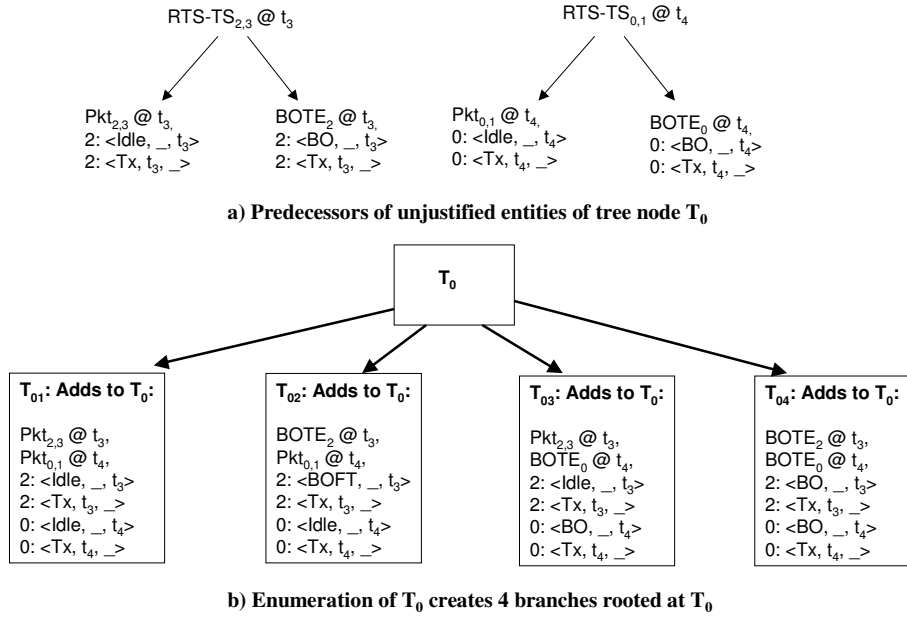


Figure 7: Enumerating child nodes.

Figure 7.(a) presents the predecessors of these unjustified entities as returned by **create-pred** procedure. Note from the transition table in Table 3 that an RTS-TS has two predecessors: predecessor in row 1 represents a transmission of RTS when the node receives a packet from higher layer, and predecessor in row 9 represents a retransmission of RTS. Each RTS-TS events in the scenario in T_0 has two predecessors, cross product of which results in 4 child nodes: T_{01} , T_{02} , T_{03} , and T_{04} presented in Figure 7.(b). Consider the tree node T_{01} presented in Figure 8. The node is a fully specified scenario as all entities in the scenario are justified, and hence it is a leaf node. A leaf node presents a output scenario that leads to the target error description.

Figure 9 presents the timing diagram of the scenario. Note that using this timing diagram we can regenerate the output test scenario that leads to the target error and simulate the scenario in a network simulator, e.g., ns-2 [24].

4 Case Study: IEEE 802.11

IEEE 802.11 is based on single channel multiple access schemes. The channel access is based on both physical and virtual carrier sensing mechanisms. [11] specifies the protocol. In this study, we only consider the protocol mechanisms in DCF (Distributed Coordinated Function). DCF is based on *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA). It allows channel access when both of the channel sensing mechanisms indicates the channel idle. Each node maintains a counter or timer called *Network Allocation Vector* (NAV) in order to monitor the channel status reserved by other nodes it hears. When node 0 of topology I wants to transmit data to node 1 , it first senses the channel status. If the channel is idle for a predefined period (DIFS), it sends RTS to 1 indicating the period T which it expects to reserve the channel if RTS/CTS is successful. Node 0 schedules a wait-for-CTS (WCTS) timer to limit the period of time to wait for CTS. When node 1 receives RTS, if its NAV timer is not running, it immediately replies with a CTS packet in the next frame slot. Otherwise,

T_{01}	H_g:	H_s:	SOI:	PL_s:
	RTS-RE _{2,3} @ t ₀ ,	1: <BOCOL, t ₂ , ->	t ₁ = t ₀ - α _c	PL_g:
	RTS-RS _{2,3} @ t ₁ ,	2: <Tx, t ₃ , t ₅ >	d < α _r	CTS-RS _{2,3} @ τ ₀
	RTS-RS _{0,1} @ t ₂	0: <Tx, t ₄ , t ₆ >	α _r < α _c	CTS-RE _{2,3} @ τ ₁
	RTS-TS _{2,3} @ t ₃ ,	2: <WCTS, t ₅ , ->	t ₁ ≤ t ₂	PL_{soi}:
	RTS-TS _{0,1} @ t ₄ ,	0: <WCTS, t ₆ , ->	t ₂ ≤ t ₀	t ₁ ≤ τ ₀ ≤ t ₀
	RTS-TE _{2,3} @ t ₅ ,	1: <Rx, t ₁ , t ₂ >	t ₃ = t ₁ - d	t ₁ ≤ τ ₁ ≤ t ₀
	RTS-TE _{0,1} @ t ₆	2: <Idle, -, t ₃ >	t ₄ = t ₂ - d	
	WCTST-TS ₂ @ t ₅ ,	0: <Idle, -, t ₄ >	t ₅ = t ₃ + α _r	
	WCTST-TS ₀ @ t ₆		t ₆ = t ₅ + α _r	
Pkt _{2,3} @ t ₃ ,				
Pkt _{0,1} @ t ₄ ,				

Figure 8: A Leaf node: output scenario leading to target error in Figure 5.(a).

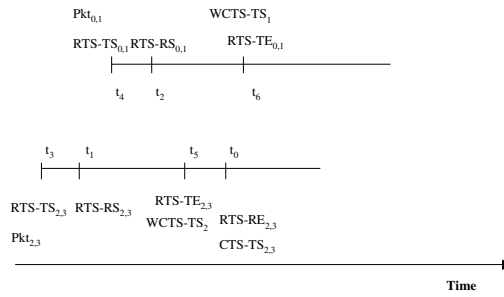


Figure 9: Timing diagram of scenario at leaf node in Figure 8.

it silently discards RTS packet. While sending CTS, node 1 schedules a wait-for-data (WDATA) timer to limit the period of time to wait for data. All other nodes in the range of transmitter and/or receiver update their NAV with the value as given in the RTS/CTS packets. During the period when a node has NAV running, it defers access to the channel. When node θ receives CTS before its WCTS timer expires, it immediately transmits data while scheduling wait-for-ACK (WACK) timer to wait for ACK. If ACK from node 1 does not arrive within this period, node θ backs off and contends for the channel again. Upon reception of data from node θ , node 1 immediately sends an ACK back to node θ . RTS/CTS exchange is aimed at reducing the probability of collision due to hidden and exposed terminals and improving throughputs. Table 8 presents some protocol events, states and time variables. Table 9 presents a part of the transition table used in our study. The set of states, events, and time variables of IEEE 802.11 is a superset of those in the example protocol P presented in Section 3.

4.1 Assumptions

The basic assumptions of the case studies are as follows. First, we assume the network is static, i.e. neighborhood of a node is fixed and does not change. Second, we do not model message loss explicitly, however, loss implied by collision and silent drop is implicitly modeled. Third, when a receiving node i in the neighborhood of two transmitting nodes j and k receives from both, it *always* leads to a *collision*, and never leads to a *capture*. When a receiver in the neighborhood of two transmitters receiving transmissions from the nodes is unable to cleanly receive signal from either nodes, the phenomenon is known as *collision*. When the receiver is able to cleanly receive the signal from closer transmitter, the phenomenon is called *capture*. We model collision, not capture. In the current version of our framework, we only model virtual carrier sense mechanisms. In this case study, we consider two types of error descriptions: 1) collision, and 2) unnecessary defer. Collision is described as overlapping of two or more reception at a node. A node i is said to be deferring unnecessarily during an interval $[t_0, t_1]$ if there is no transmission during the interval which was reserved for a future transmission.

Table 8: Subset of protocol events, states and time variables of IEEE 802.11.

Event	Description
Data-TS _{i,j}	Data-transmission-start from i to j
Data-TE _{i,j}	Data-transmission-end from i to j
Data-RS _{i,j}	Data-reception-start from i to j
Data-RE _{i,j}	Data-reception-end from i to j
State	Description
Idle	Idle state
Tx	Transmitting
Rx	Receiving
WCTS	Waiting for Clear-to-transmit
WCTSAR	Waiting for CTS and receiving
Time variables	Description
α_r	Time to transmit/receive RTS
α_c	Time to transmit/receive CTS
β	Time to transmit/receive Data
d	Propagation delay
θ_r	Defer period on RTS
θ_c	Defer period on CTS
w_c	WCTS period
B_c	Backoff period

Table 9: Part of Transition table.

Start state	Input event	End state	Output event
Idle _i	Pkt _i	Tx _i	RTS _{i,j} -TS(0)[i], RTS _{i,j} -RS(d)[G _i], RTS _{i,j} -TE(α_r)[i], RTS _{i,j} -RE(d+ α_r)[G _i]
Tx _i	RTS _{i,j} -TE	WCTS _i	CTST-S(0)[i], CTST-E($\alpha_c + \delta + 2d$)[i]
WCTS _i	CTS _{i,j} -RS	WCTSAR _i	
WCTSAR _i	CTS _{i,j} -RE	WSIFS-C _i	SIFST-S(0)[i], SIFST-E(δ)[i]
WSIFS-C _i	SIFST-E _i	Tx _i	DATA _{i,j} -TS(0)[i], DATA _{i,j} -RS(d)[G _i], DATA _{i,j} -TE(β)[i], DATA _{i,j} -RE(d+ β)[G _i]

4.2 Examples

We used our framework to analyze two types of errors in the IEEE 802.11 protocol: (1) collision, and (2) unnecessary defer. We illustrate 3 example scenarios in this section for the topology in Figure 1. The first example illustrates our pruning algorithm, and the other two examples illustrate output scenarios that lead to the target error description.

Example 1: An invalid scenario: Figure 10 presents an invalid scenario that is pruned at the tree root T_0 . Figure 10.(a) presents the error description that describes a collision of CTS_{2,3} and CTS_{0,1} at node 1 when the length of RTS and CTS messages are equal. Implication of CTS-RS_{2,3} generates CTS-TS_{2,3} (Figure (b)) which generates CTS-TE_{2,3} and RTS-RE_{3,2} (Figure (c)) in the scenario. RTS-RE_{3,2} indicates a successful reception at node 2, implication of which by Lemma 5 indicates that no node from which node 2 hears should not transmit during the reception interval interval $[t_7, t_3]$. The procedure **check-existence** finds an exact match of the prohibited list entry RTS-RE_{1,0} at τ_1 with H_e entry RTS-RE_{1,0} at t_6 . At this point, the scenario is pruned as presented in Figure 10.(d). This example demonstrates the usefulness and efficiency of our implication rules in identifying invalid branches and pruning them very early in the search process (in the tree root in this example).

Example 2: Valid collision scenarios: Figure 11.(a) presents a collision description of CTS_{2,3} and CTS_{0,1} at node 1 when the length of RTS is less than that of a CTS message. Figure 12 presents the scenario in T_0 after 7 rounds of implications at which point no new information is generated. Note that the relation ($t_3 < t_8$) in *SOI* in the figure guarantees that RTS_{3,2} and RTS_{1,0} do not collide at node 2. This time relation is generated by successful reception of RTS_{3,2} at node 2. The unjustified entities of the tree is presented in **bold** which are enumerated and 4 child nodes are created as presented in Figure 13. The child node T_0 represents the case where no retransmission is involved, whereas the rest 3 child nodes involve retransmission of one or two of the RTSs. Figure 14 presents the timing diagram of scenario S_{C_1} described in tree leaf node T_{01} . The timing diagram is generated manually using the time relations of the scenario. Note that the relation ($t_3 < t_8$) is necessary for node 2 to successfully receive RTS_{3,2}. If this relation is not maintained, RTS_{3,2} will collide with RTS_{1,0} at node 2 resulting an unsuccessful reception of RTS_{3,2}. Following is a library of all valid scenarios that are output of our framework.

- **Sc1c₁**: The scenario is presented in Figure 14. The scenario does not involved any retransmission. All other scenarios in the library involves retransmission of RTS_{3,2} the response to which generates the CTS_{2,3} on target collision.
- **Sc1c₂**: The scenario is rooted at the node T_{01} . The node 2 defers on a RTS_{1,0}, and silently drops RTS_{3,2}. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$. The target CTS_{2,3} is a response to retransmitted RTS_{3,2}.
- **Sc1c₃**: In this scenario, the node 2 defers on a CTS_{1,0}, and silently drops RTS_{3,2}. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$.
- **Sc1c₄**: The node 2 silently drops RTS_{3,2} as it was on back-off state on collision of previous incarnation of RTS_{3,2} and RTS_{1,0}. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$.

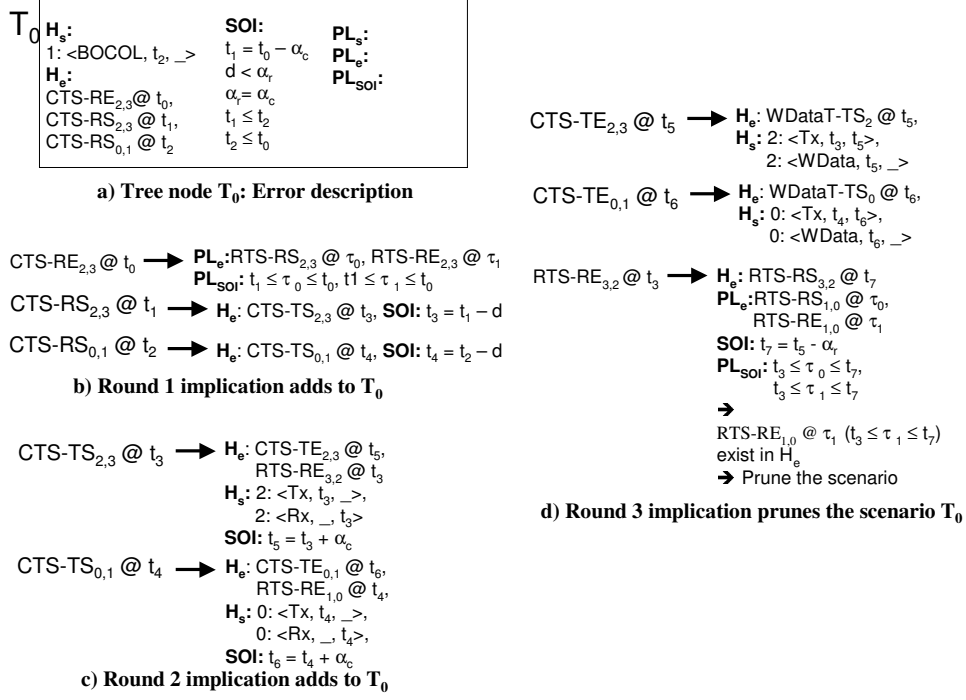


Figure 10: An invalid scenario.

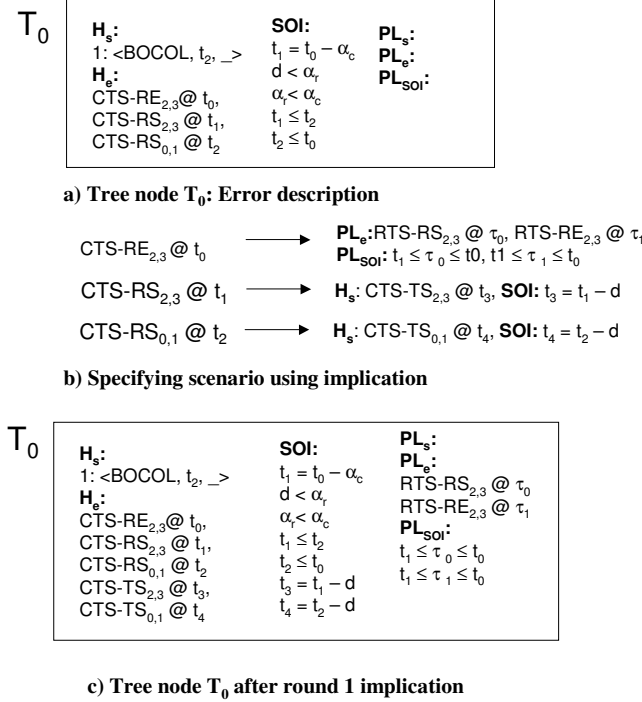


Figure 11: A collision description of IEEE 802.11.

T_0	H_e :	H_s :	SOI:	PL_s : PL_e :	PL_{SOI} :
	CTS-RE _{2,3} @ t_0	1: <BOCOL, t_2 , $_$ >	$t_1 = t_0 - \alpha_c$	RTS-RS _{2,3} @ τ_0	$t_1 \leq \tau_0 \leq t_0$
	CTS-RS _{2,3} @ t_1	2: <TX, t_3 , t_5 >	$d < \alpha_r$	RTS-RE _{2,3} @ τ_1	$t_1 \leq \tau_1 \leq t_0$
	CTS-RS _{0,1} @ t_2	2: <RX, t_7 , t_3 >	$\alpha_r < \alpha_c$	RTS-RS _{1,0} @ τ_2	$t_7 \leq \tau_2 \leq t_3$
	CTS-TS _{2,3} @ t_3	0: <TX, t_4 , t_5 >	$t_1 \leq t_2$	RTS-RE _{1,0} @ τ_3	$t_7 \leq \tau_3 \leq t_3$
	CTS-TS _{0,1} @ t_4	0: <RX, t_3 , t_4 >	$t_2 \leq t_0$	CTS-RS _{1,0} @ τ_4	$t_7 \leq \tau_4 \leq t_3$
	CTS-TE _{2,3} @ t_5	2: <Idle, t_5 , $_$ >	$t_3 = t_1 - d$	CTS-RE _{1,0} @ τ_5	$t_7 \leq \tau_5 \leq t_3$
	CTS-TE _{0,1} @ t_6	0: <Idle, t_6 , $_$ >	$t_4 = t_2 - d$	RTS-RS _{1,2} @ τ_6	$t_7 \leq \tau_6 \leq t_3$
	RTS-RE _{3,2} @ t_3	3: <TX, t_9 , t_{11} >	$t_5 = t_3 + \alpha_c$	RTS-RE _{1,2} @ τ_7	$t_7 \leq \tau_7 \leq t_3$
	RTS-RE _{1,0} @ t_4	1: <TX, t_{10} , t_{12} >	$t_6 = t_4 + \alpha_c$	CTS-RS _{1,2} @ τ_8	$t_7 \leq \tau_8 \leq t_3$
	RTS-RS _{3,2} @ t_7	3: <WCTS, t_{11} , $_$ >	$t_7 = t_3 - \alpha_r$	CTS-RE _{1,2} @ τ_9	$t_7 \leq \tau_9 \leq t_3$
	RTS-RS _{1,0} @ t_8	1: <WCTS, t_{12} , $_$ >	$t_8 = t_4 - \alpha_r$	CTS-RS _{3,2} @ τ_{10}	$t_7 \leq \tau_{10} \leq t_3$
	RTS-TS _{3,2} @ t_9	3: <Idle, $_$, t_9 >	$t_9 = t_7 - d$	CTS-RE _{3,2} @ τ_{11}	$t_7 \leq \tau_{11} \leq t_3$
	RTS-TS _{1,0} @ t_{10}	1: <Idle, $_$, t_{10} >	$t_{10} = t_8 - d$		
	RTS-TE _{3,2} @ t_{11}		$t_3 < t_8$		
	RTS-TE _{1,0} @ t_{12}		$t_{11} = t_9 + \alpha_r$		
	WCTST-TS ₃ @ t_{11}		$t_{12} = t_{10} + \alpha_r$		
	WCTST-TS ₁ @ t_{12}				

Figure 12: The scenario after 7 rounds of implications.

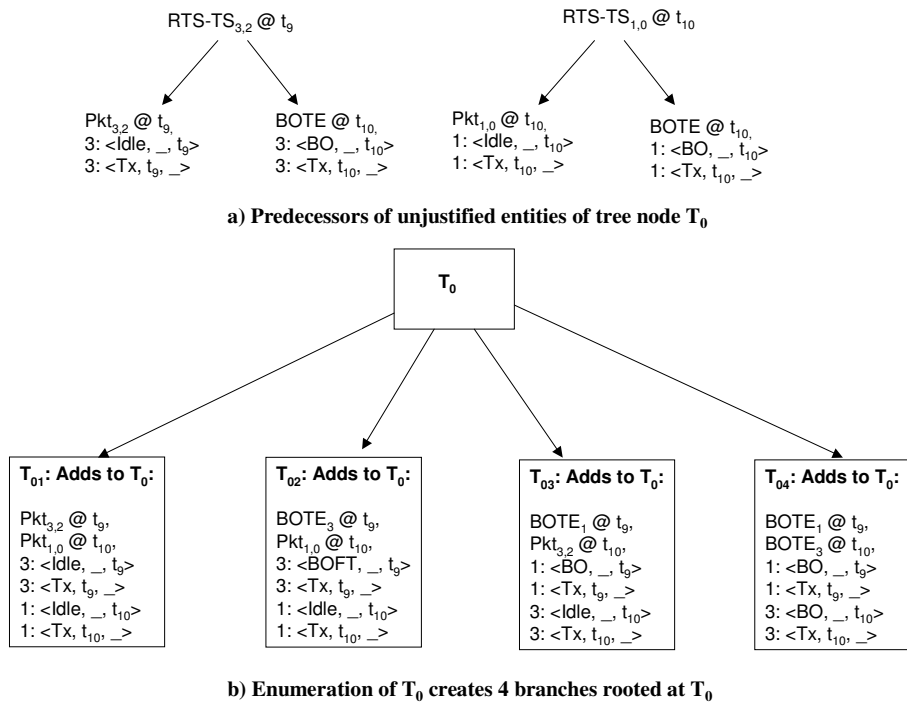


Figure 13: Enumerating child nodes.

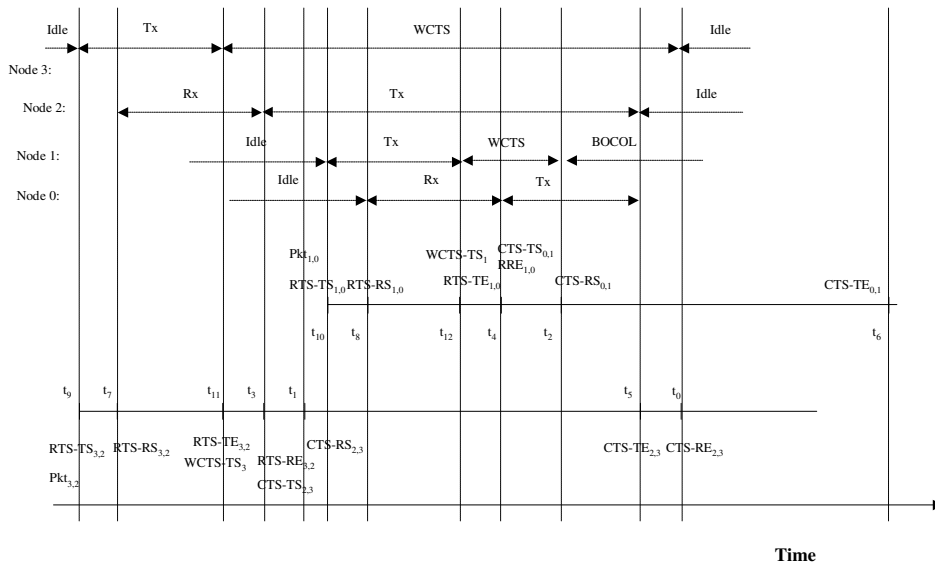


Figure 14: Timing diagram of scenario in leaf node T_{01} .

- **Sc1c₅**: The node 2 silently drops RTS_{3,2} as it was on back-off state on collision of previous incarnation of RTS_{3,2} and CTS_{1,0}. The node 3, on expiration of back-off timer, retransmits the RTS_{3,2}.
- **Sc1c₆**: The node 2 silently drops RTS_{3,2} as it was on back-off state on collision of previous incarnation of RTS_{3,2} and Data_{1,0}. The node 3, on expiration of back-off timer, retransmits the RTS_{3,2}.
- **Sc1c₇**: The node 2 silently drops RTS_{3,2} as it was on back-off state on collision of previous incarnation of RTS_{3,2} and ACK_{1,0}. The node 3, on expiration of back-off timer, retransmits the RTS_{3,2}.
- **Sc1c₈**: The first RTS_{3,2} collides with RTS_{1,0} at node 2, and hence the node 2 does not receive it successfully. The node 3, on expiration of back-off timer, retransmits the RTS_{3,2}. The time relation generated by implication rules indicates that the node 2 must get out of the back-off state before it receives the retransmitted RTS_{3,2} so that it responds to the message with the target CTS_{2,3}.
- **Sc1c₉**: The RTS_{3,2} collides with CTS_{1,0} at node 2, and hence the node 2 does not receive it successfully. The node 3, on expiration of back-off timer, retransmits the RTS_{3,2}. The time relation generated by implication rules indicates that the node 2 must get out of the back-off state before it receives the retransmitted RTS_{3,2} so that it responds to the message with the target CTS_{2,3}.
- **Sc1c₁₀**: The RTS_{3,2} collides with Data_{1,0} at node 2, and hence the node 2 does not receive it successfully. The node 3, on expiration of back-off timer, retransmits the RTS_{3,2}. The time relation generated by implication rules indicates that the node 2 must get out of the back-off state before it receives the retransmitted RTS_{3,2} so that it responds to the message with the target CTS_{2,3}.
- **Sc1c₁₁**: The RTS_{3,2} collides with ACK_{1,0} at node 2, and hence the node 2 does not receive it successfully. The node 3, on expiration of back-off timer, retransmits the RTS_{3,2}. The time relation generated by implication rules indicates that the node 2 must get out of the back-off state before it receives the retransmitted RTS_{3,2} so that it responds to the message with the target CTS_{2,3}.

Note that the scenarios **Sc1c₈** thru **Sc1c₁₁** are direct second order collision scenarios in the sense that the target collision (primary) is a consequence of an earlier collision in the network. The scenarios **Sc1c₄** thru **Sc1c₇** are indirect second order collision scenarios in the sense that the target collision is an indirect consequence of an earlier collision in the network. The direct second order collision will occur without further timing constraints while the indirect collision are subject to other timing constraints, e.g. values of back-off timers etc.

Example 3: Valid unnecessary defer scenarios: A node is said to be deferring unnecessarily for a period $[t_0, t_1]$ if there is no transmission during the interval for which the channel was reserved and the node is deferring. Figure 15.(b) presents an error description where node 4 in Figure 15 is deferring for a period $[t_0, t_1]$ on overhearing RTS-RS_{3,2} at t_2 . From the time relation ($t_2 = t_0 - \alpha_r$), it is ensured in the description that the node 4 starts deferring once it has received the RTS_{3,2}. Note that the prohibited entries in the figure ensures that there should be no data transmission from node 3 to node 2 during $[t_0, t_1]$ interval. Figure 16 presents one of the output scenarios generated that leads to the target scenario. In this scenario, node 2 is not responding to RTS_{3,2} destined for it as it is deferring for a period $[t_6, t_7]$ on overhearing the RTS_{1,0}. The node 4 defers on overheard RTS_{3,2} but there is no Data_{3,2} transmission during the period $[t_0, t_1]$ as RTS_{3,2} was silently discarded by node 2. Note that the time relation ($t_6 \leq t_2 \leq t_7$) generated by the algorithm guarantees the node 2 to silently drop the RTS_{3,2}. Following is a library of scenarios generated by our framework leading to the target unnecessary defer.

- **Sc1d₁**: The scenario presented in Figure 15 in which node 2 defers on overheard RTS_{1,0} during which it silently drops the RTS_{3,2} to cause node 4 to defer unnecessarily on overheard unsuccessful RTS_{3,2}.
- **Sc1d₂**: Node 2 defers on overheard CTS_{1,0} during which it silently drops the RTS_{3,2}.
- **Sc1d₃**: Node 2 is in WCTS state waiting for a CTS_{1,2} during which it silently drops the RTS_{3,2}.
- **Sc1d₄**: Node 2 is in BOFT (back-off on failed transmission of RTS_{2,1}) state during which it silently drops the RTS_{3,2}.

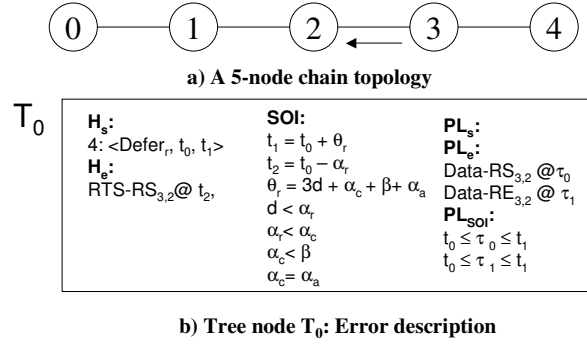


Figure 15: An unnecessary defer description of IEEE 802.11.

H_s: 4: <Defer _l , t ₀ , t ₁ > 3: <Tx, t ₃ , t ₄ > 3: <WCTS, t ₄ , -> 4: <Rx, -, t ₆ > 3: <Idle, -, t ₃ > 2: <Defer _r , t ₆ , t ₇ > 2: <Rx, -, t ₆ > 1: <Tx, t ₉ , t ₁₁ > 2: <Idle, t ₁₀ , -> 1: <WCTS, t ₁₁ , -> 1: <Idle, -, t ₉ >	H_e: RTS-RS _{3,2} @t ₂ RTS-TS _{3,2} @t ₃ RTS-TE _{3,2} @t ₄ WCTST-TS ₃ @t ₄ RTS-RE _{3,2} @t ₀ Pkt _{3,2} @t ₃ DeferT1-TS ₄ @t ₀ DeferT1-TE ₄ @t ₃ RTS-RE _{1,0} @t ₆ RTS-RS _{1,0} @t ₈ DeferT1-TS ₂ @t ₆ RTS-TS _{1,0} @t ₉ DeferT1-TE ₂ @t ₁₀ RTS-TE _{1,0} @t ₁₁ WCTST-TS ₁ @t ₁₁ Pkt _{1,0} @t ₉	SOI: t ₁ = t ₀ + θ _r t ₂ = t ₀ - α _r θ _r = 3d + α _c + β + α _a d < α _r α _r < α _c α _c < β α _c = α _a t ₃ = t ₂ - d t ₄ = t ₃ + α _r t ₅ = t ₀ + θ _r t ₆ < t ₂ t ₂ < t ₇ t ₈ = t ₆ - α _r t ₉ = t ₈ - d t ₁₀ = t ₆ + θ _r t ₁₁ = t ₉ + α _r t ₇ = t ₁₀	PL_s: PL_e: Data-RS _{3,2} @τ ₀ Data-RE _{3,2} @τ ₁ PL_{soi}: t ₀ ≤ τ ₀ ≤ t ₁ t ₀ ≤ τ ₁ ≤ t ₁
---	--	--	--

Figure 16: An output scenario leading to target error in Figure 15.

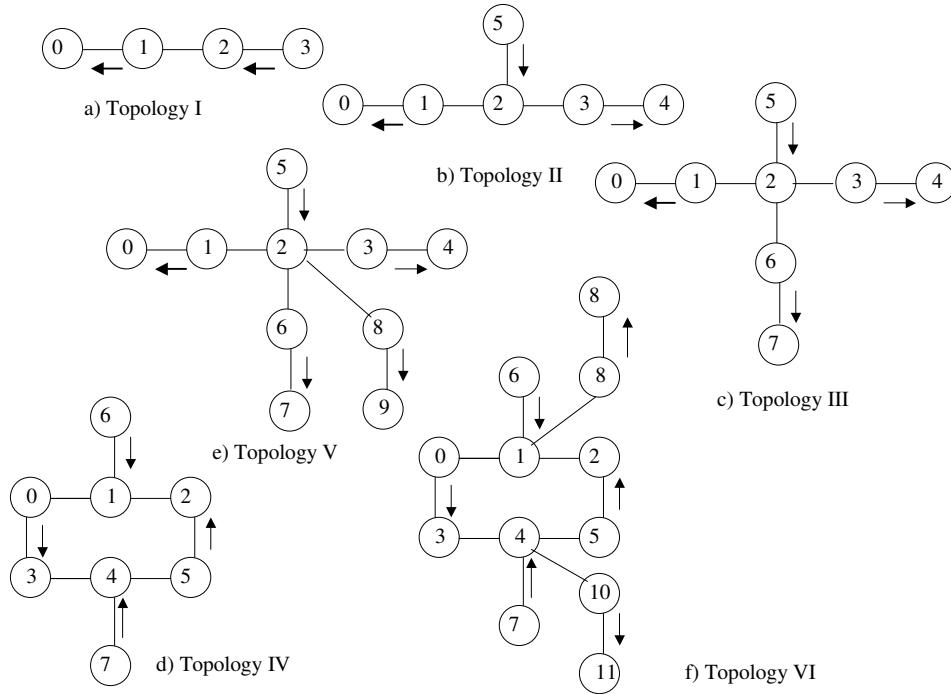


Figure 17: Network topologies used in simulation.

- **Sc1d₅**: Node 2 is in BOCOL (back-off on collision of previous $RTS_{3,2}$ and $RTS_{1,0}$) state during which it silently drops the $RTS_{3,2}$.
- **Sc1d₆**: Node 2 is in BOCOL (back-off on collision of previous $RTS_{3,2}$ and $CTS_{1,0}$) state during which it silently drops the $RTS_{3,2}$.
- **Sc1d₇**: Node 2 is in BOCOL (back-off on collision of previous $RTS_{3,2}$ and $Data_{1,0}$) state during which it silently drops the $RTS_{3,2}$.
- **Sc1d₈**: Node 2 is in BOCOL (back-off on collision of previous $RTS_{3,2}$ and $ACK_{1,0}$) state during which it silently drops the $RTS_{3,2}$.

4.3 Simulation Results

Given an error description for a topology, we first use our framework to generate sequence of protocol events that lead to the target error. The objective of simulating these output scenarios is: 1) the regenerated scenarios in simulation framework verify our results, and 2) using these scenarios we can analyze the performance of the protocol. We reproduced collision and unnecessary defer scenarios in ns-2 simulator, however, we only used the unnecessary defer scenarios to analyze the performance of protocol.

We run simulations of EOTG scenarios on topologies shown in Figure 17 in ns-2 simulator. The arrows in the figure represent the directions of flows in the scenarios. We use CBR sources at a rate of 6 MBPS for topologies III, IV, VI, and 0.6 MBPS for the rest. We regenerate the sequence of CBR sources according to the test scenarios generated from our algorithm. For example, in Figure 17.(a) the start times of $1 \rightarrow 0$ flow and $3 \rightarrow 2$ flow are 5 seconds and 5.01, seconds respectively. Total simulation time is 50 seconds for all simulations. Figure 18 presents the throughput of destination nodes of individual flows and total network throughput for each of the

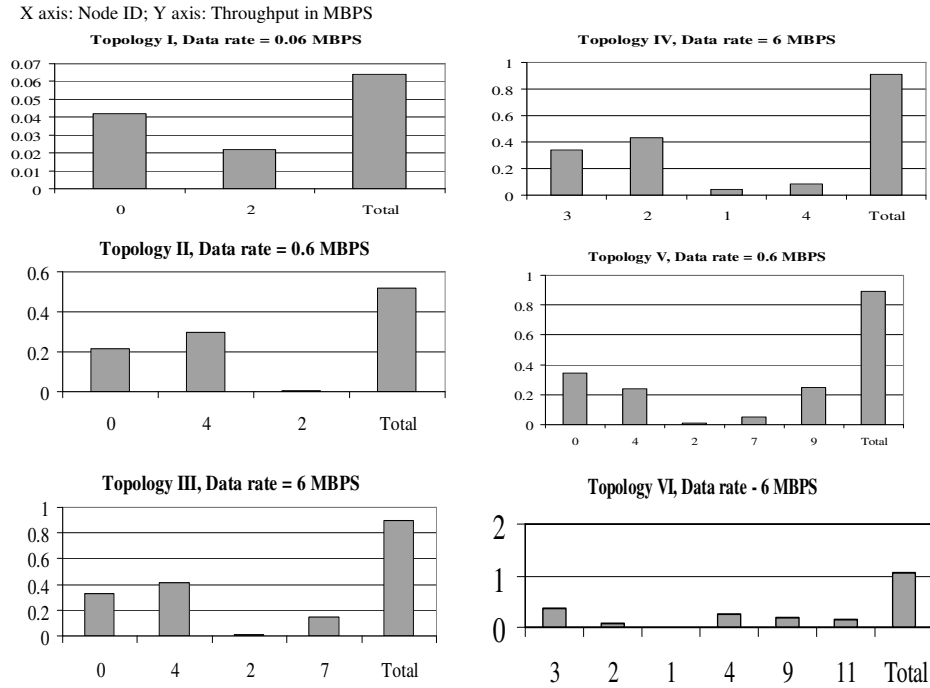


Figure 18: Average throughput of networks in Figure 17.

topologies presented in Figure 17. Note that in Figure 18.(a), the average throughput of node 2 is about 50% of the average throughput of node 0. Based on the basic topology of unnecessary defer scenario, we systematically construct topologies with two objectives: (1) to allow a target node to starve more, and/or (2) to allow more nodes to starve. For example in topology II (Figure 17.(b)), defer state of node 2 is extended because of the transmission from $1 \rightarrow 0$ and $3 \rightarrow 4$ resulting in almost zero throughput. Topologies III and IV are extensions where more nodes starve as shown in Figure 18. Topologies V and VI are extensions from topologies III and IV, respectively in which throughput of one of the nodes reaches almost zero. Note that in all these scenarios, all other nodes achieve average throughput except the target nodes. These results demonstrate that IEEE 802.11 is unfair in a sense that some nodes in the network starve completely while other nodes achieve average throughput. It also demonstrates that the throughput can reach to zero with the increase of number of ongoing transmissions in the neighborhood. Such short term unfairness severely affect performance of TCP and real-time applications [20]. Note that channel utilization of topologies III and IV is **3.7%** and that of topology VI is **2.9%**, which are very low compared to 45-65% typically obtained in previous studies [22] of performance evaluation.

4.4 Performance Enhancement using Test Generation Results

From the library of test scenarios, we found that the silent drop of RTS by a node in certain states (e.g., defer, backoff) is a leading cause of these errors. We then modified the protocol by introducing a receiver initiated approach such that a node, instead of silent drop, initiates a request to the sender to transmit the RTS it received on defer (and backoff). With this modification, there was no unnecessary defer scenarios, however, it introduced some collision scenarios.

5 Case Study: MACAW

MACAW (Multiple Access Collision Avoidance Wireless) [13] is derived from MACA [12] for the unreliability of wireless medium. The basic handshaking mechanisms of MACAW are similar to IEEE 802.11 with the following differences. First, there is no physical carrier sense in MACAW. Second, a node, after receiving a CTS in response to its RTS message, sends a short DS (Data-Send) packet before sending the data. The DS packet allows the neighborhood of the transmitter to be aware of the successful RTS/CTS handshake. Third, while deferring access to the channel on overheard RTS/CTS, a node does not drop an RTS destined for it. During the next contention period, the node sends out an Request for RTS (RRTS) message to the sender of the RTS to initiate the RTS/CTS handshake. For this case study, we first model the MACAW protocol using a transition table. We take the same topology and error description as the case study of IEEE 802.11. We then apply our algorithms to generate test scenarios that lead to the error in MACAW. The assumptions of the case study of MACAW is same as those presented in Section 4.1 for our case study of IEEE 802.11 protocol. In this section, we present a comparative analysis of MACAW and IEEE 802.11 with respect to the test scenarios generated. Section 5.1 presents example test scenarios for similar collision and unnecessary defer scenarios presented in the case study of IEEE 802.11 in Section 4.2. Section 5.2 presents the comparison of the protocols.

5.1 Examples

Example 1: Valid collision scenarios: We refer to Figure 11.(a) that presents a collision description of $CTS_{2,3}$ and $CTS_{0,1}$ at node 1 when the length of RTS is less than that of a CTS message. Following is a library of all valid scenarios generated using our framework.

- **Sc2c₁:** The scenario is same as IEEE 802.11 presented in Figure 14. The scenario does not involved any retransmission. Note that the length of protocol messages and timer variables depend on the protocol, hence may result in a different sequence in different protocols. Scenarios **Sc2c₂** thru **Sc2c₂** involves retransmission of $RTS_{3,2}$.
- **Sc2c₂:** The node 2 silently drops $RTS_{3,2}$ as it was on back-off state on collision of previous incarnation of $RTS_{3,2}$ and $RTS_{1,0}$. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$.
- **Sc2c₃:** The node 2 silently drops $RTS_{3,2}$ as it was on back-off state on collision of previous incarnation of $RTS_{3,2}$ and $CTS_{1,0}$. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$.
- **Sc2c₄:** The node 2 silently drops $RTS_{3,2}$ as it was on back-off state on collision of previous incarnation of $RTS_{3,2}$ and $DS_{1,0}$. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$.
- **Sc2c₅:** The node 2 silently drops $RTS_{3,2}$ as it was on back-off state on collision of previous incarnation of $RTS_{3,2}$ and $Data_{1,0}$. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$.
- **Sc2c₆:** The node 2 silently drops $RTS_{3,2}$ as it was on back-off state on collision of previous incarnation of $RTS_{3,2}$ and $ACK_{1,0}$. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$.
- **Sc2c₇:** The node 2 silently drops $RTS_{3,2}$ as it was on back-off state on collision of previous incarnation of $RTS_{3,2}$ and $RRTS_{1,0}$. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$.
- **Sc2c₈:** The first $RTS_{3,2}$ collides with $RTS_{1,0}$ at node 2, and hence the node 2 does not receive it successfully. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$. The time relation generated by implication rules indicates that the node 2 must get out of the back-off state before it receives the retransmitted $RTS_{3,2}$ so that it responds to the message with the target $CTS_{2,3}$.
- **Sc2c₉:** The $RTS_{3,2}$ collide with $CTS_{1,0}$ at node 2, and hence the node 2 does not receive it successfully. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$. The time relation generated by implication rules indicates that the node 2 must get out of the back-off state before it receives the retransmitted $RTS_{3,2}$ so that it responds to the message with the target $CTS_{2,3}$.

- **Sc2c₁₀**: The $RTS_{3,2}$ collide with $DS_{1,0}$ at node 2, and hence the node 2 does not receive it successfully. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$. The time relation generated by implication rules indicates that the node 2 must get out of the back-off state before it receives the retransmitted $RTS_{3,2}$ so that it responds to the message with the target $CTS_{2,3}$.
- **Sc2c₁₁**: The $RTS_{3,2}$ collide with $Data_{1,0}$ at node 2, and hence the node 2 does not receive it successfully. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$. The time relation generated by implication rules indicates that the node 2 must get out of the back-off state before it receives the retransmitted $RTS_{3,2}$ so that it responds to the message with the target $CTS_{2,3}$.
- **Sc2c₁₂**: The $RTS_{3,2}$ collide with $ACK_{1,0}$ at node 2, and hence the node 2 does not receive it successfully. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$. The time relation generated by implication rules indicates that the node 2 must get out of the back-off state before it receives the retransmitted $RTS_{3,2}$ so that it responds to the message with the target $CTS_{2,3}$.
- **Sc2c₁₃**: The $RTS_{3,2}$ collide with $RRTS_{1,0}$ at node 2, and hence the node 2 does not receive it successfully. The node 3, on expiration of back-off timer, retransmits the $RTS_{3,2}$. The time relation generated by implication rules indicates that the node 2 must get out of the back-off state before it receives the retransmitted $RTS_{3,2}$ so that it responds to the message with the target $CTS_{2,3}$.

Note that scenarios **Sc2c₈** thru **Sc2c₁₃** are direct and **Sc2c₂** thru **Sc2c₇** are indirect second order collision scenarios.

Example 2: Valid unnecessary defer scenarios: We refer to Figure 15.(b) that presents an error description where node 4 in Figure 15 is deferring for a period $[t_0, t_1]$ on overhearing RTS - $RS_{3,2}$ at t_2 . Following is a library of scenarios.

- **Sc2d₁**: Node 2 is in WCTS state waiting for a $CTS_{1,2}$ during which it silently drops the $RTS_{3,2}$.
- **Sc2d₂**: Node 2 is in BOFT (back-off on failed transmission of $RTS_{2,1}$) state during which it silently drops the $RTS_{3,2}$.
- **Sc2d₃**: Node 2 is in BOCOL (back-off on collision of previous $RTS_{3,2}$ and $RTS_{1,0}$) state during which it silently drops the $RTS_{3,2}$.
- **Sc2d₄**: Node 2 is in BOCOL (back-off on collision of previous $RTS_{3,2}$ and $CTS_{1,0}$) state during which it silently drops the $RTS_{3,2}$.
- **Sc2d₅**: Node 2 is in BOCOL (back-off on collision of previous $RTS_{3,2}$ and $DS_{1,0}$) state during which it silently drops the $RTS_{3,2}$.
- **Sc2d₆**: Node 2 is in BOCOL (back-off on collision of previous $RTS_{3,2}$ and $Data_{1,0}$) state during which it silently drops the $RTS_{3,2}$.
- **Sc2d₇**: Node 2 is in BOCOL (back-off on collision of previous $RTS_{3,2}$ and $ACK_{1,0}$) state during which it silently drops the $RTS_{3,2}$.
- **Sc2d₈**: Node 2 is in BOCOL (back-off on collision of previous $RTS_{3,2}$ and $RRTS_{1,0}$) state during which it silently drops the $RTS_{3,2}$.

5.2 Comparison of MACAW and IEEE 802.11

We present a comparative analysis of IEEE 802.11 and MACAW in this section based on the test results presented in Sections 4.2 and 5.1 respectively. In IEEE 802.11, a node silently drops RTS destined for it when it defers access to channel, while in MACAW the node does not drop the RTS. Instead it initiates the handshake by sending an RRTS message to the sender of the RTS. Note that in MACAW, there are no collision and defer scenarios where a node silently drops RTS while on defer state. Further, a successful RTS/CTS exchange in IEEE 802.11 is not confirmed on the neighborhood of transmitter, while it is confirmed in MACAW with the use of DS packet.

These two additional features remove some collision and unnecessary defer scenarios in MACAW, however, these features add more control messages resulting in more scenarios that involve collision of control packets.

6 Analysis of the Framework

6.1 Correctness and Completeness

Given a partial scenario at a tree node T_n , Lemma 2 guarantees that we enumerate all child nodes rooted at T_n . Given an error description as partial scenario, we first copy the description to the root of the search tree. We then apply a DFS search technique starting from the root. The search and implication continue until we (1) reach at a leaf node, or (2) we prune the scenario based on our implication rules. If we show that our algorithm detects a leaf node correctly, and prunes a tree node correctly, we can guarantee the completeness of our algorithm in the sense that it will always generate all valid scenarios that lead to the target error. Lemma 3 shows that our algorithm must stop enumeration at a leaf node that ensures the generation of valid scenarios. Lemmas 5, 6, and 7 show that the algorithm detects the existence of prohibited event entries in the scenario, while Lemma 8 detects any inconsistency in the state history. Hence the algorithm correctly identifies invalid scenarios to prune. Therefore, the correctness of our algorithm guarantees its completeness.

6.2 Complexity

We present a brief analysis of our test generation results to provide a basic idea of complexity of our search and implication algorithms using the case study of IEEE 802.11. Note that we use implication to derive node state and event history, and to eliminate invalid branches. We can generate valid scenario using only our search algorithm, however, without implication algorithm we will not be able to detect any invalid scenario. For this reason, we use implication algorithm in our case study. Implication reduces the complexity of search. Example 2 in Section 4.2 enumerates 4 child nodes after 7 rounds of implication. Without performing the implication, it would enumerate 12 child nodes at tree root, that would increase exponentially as it goes down the tree. Table 11 presents statistics recorded from output of our framework under different types of error on different topologies. Total number of states, events, and rows in transition table of IEEE 802.11 are 15, 31, and 156, respectively. The first scenario is an invalid scenario and the rest are valid scenarios. Scenario CCC1 and CCC2 are CTS-CTS collision on chain and non-linear topologies respectively. DDC is a Data-Data collision scenario. The column denoted by **Nodes** indicates the number of nodes in the topology while the column denoted by **Deg.** indicates the maximum node degree of the network. The column denoted by **Hgt.** indicates the height of the search tree at which the scenarios was generated (or, pruned in case of invalid scenario). The column denoted by **I.Round**, **I.Call**, **exist**, and **LP** indicate the total rounds of implication, the total number of implication calls, total calls of **check-exist** procedure, and total calls of LP respectively. Note that the complexity of our algorithm depends on the following: (1) the size of transition table, (2) topology, and (3) the target error description.

Table 10: Search and implication statistics of various scenarios in IEEE 802.11.

Scenario	Nodes	Deg.	Hgt.	I.Round	I.call	exist	LP
Invalid	4	2	0	7	223	797	1772
CCC1	4	2	1	8	324	622	1385
CCC2	4	3	1	8	324	963	2132
DDC	4	2	6	11	642	1497	4416
Defer1	5	2	4	14	392	795	899
Defer2	7	2	7	21	874	1653	2101

We present a brief analysis of our test generation results to provide a basic idea of complexity of our search and implication algorithms using the case study of IEEE 802.11. Total number of states, events, and rows in transition table of the protocol are 15, 31, and 156, respectively. Note that we use implication to derive node state and event history, and to eliminate invalid branches. We can generate valid scenario using only our search algorithm,

however, without implication algorithm we will not be able to detect any invalid scenario. For this reason, we use implication algorithm in our case study. Implication reduces the complexity of search. Example 2 in Section 4.2 enumerates 4 child nodes after 7 rounds of implication. Without performing the implication, it would enumerate 12 child nodes at tree root, that would increase exponentially as it goes down the tree. Table 11 presents statistics recorded from output of our framework for different types of errors on various topologies. The first scenario is an invalid scenario and the rest are valid scenarios. Scenario CCC1 and CCC2 are CTS-CTS collision on chain and non-linear topologies, respectively. DDC is a Data-Data collision scenario. The column denoted by **N** indicates the number of nodes in the topology while the column denoted by **D** indicates the maximum node degree of the network. The columns denoted by **H** and **T** present the height of the tree and the total number of tree nodes (so far) at which the scenario was generated (or, pruned in case of invalid scenario), respectively. The columns denoted by **IR**, **IC**, **exist**, and **LP** indicate the total rounds of implications, the total number of implication calls, total calls of **check-exist** procedure, and total calls of LP (linear programming tool), respectively. Note that the complexity of our algorithm depends on the following: (1) the size of transition table, (2) topology, and (3) the target error description. Table 12 presents statistics of the search for scenarios CCC1 and DDC. The columns denoted by **T.Nd**, **Valid**, **Invalid**, and **Backtrack** present the total number of tree nodes, the total number of valid, invalid and backtracked scenarios generated, respectively. The column denoted by **E.Time** presents the total execution time of the test run on a 2.1 GHz Pentium4 machine in hours. The algorithm used is our first implementation without optimization or fine-tuning to improve its performance. We used LP tool LINDO [25] to solve the system of inequalities. Performance fine-tuning is a part of our future work. We backtrack whenever the search continues without generating a leaf node, or pruning with a retransmission limit of two in this case study. From this empirical analysis, the complexity of our implication algorithms shows to be linear in the number of nodes in the network for our practical purposes. Our initial analysis shows that the reduction of complexity of our search is primarily attributed to the use of our implication algorithms which are linear for this case study. However, the worst case complexity of our framework may be exponential in general due to the use of search and branch/bound techniques, although we have not encountered such case in our studies.

Table 11: Search and implication statistics of the case study of IEEE 802.11.

Scen.	N.	D.	H.	T.	IR	IC	exist	LP
Invalid	4	2	0	1	7	223	797	1772
CCC1	4	2	1	1	8	324	622	1385
CCC2	4	3	1	1	8	324	963	2132
DDC	4	2	6	1	11	642	1497	4416
Defer1	5	2	4	1	14	392	795	899
Defer2	7	2	7	1	21	874	1653	2101
Defer3	9	2	10	1	28	1230	3350	3909

Table 12: Search statistics of the case study of IEEE 802.11.

Scen.	T.Nd	Valid	Invalid	Backtrack	E.Time
CCC1	1636	40	904	200	7.45
DDC	464	16	174	71	10.41

6.3 Strengths and Limitations

The complexity of our framework is reduced because of the mix of backward and forward search as well as the use of implication rules to eliminate invalid choices earlier. The reduction in complexity strengthen our framework to be used to test a large topology. The library of test scenarios can be input to network simulator for performance evaluation. The case studies indicate that unfairness of IEEE 802.11 can be improved using control message, e.g., RRTS similar to MACAW, however, that would increase the number of collisions in the protocol. Thus, the most powerful application of our framework is in the design of network protocol. Protocol designers can use our framework to test extreme performance. Based on the test results, designers can identify the problems in design, modify and re-test. The main limitation of our framework is that it is not probabilistic.

7 Summary and Future Work

We developed an error oriented test generation framework for testing wireless adhoc MAC layer protocols. The framework is based on search methods which is a mix of forward and backward search. Given an error described in terms of network node states, events and time relations, our framework generates all scenarios leading to the error. We have used our framework to analyze performance of IEEE 802.11. Using it, we have generated scenarios that lead up to extreme unfairness and a throughput reduction of **90%** in the network. The empirical results show that our algorithm is quite manageable for our practical purposes. We plan to use our framework for other verification and synthesis purposes, e.g., topology synthesis, studying other classes (e.g., sensor networks) and behaviors (e.g., mobility) of protocols. In our current framework, we deal with partially specified scenarios with full information of topology which can be naturally extended to deal with partial information of topology and then synthesize the topology using a different set of implication rules.

Appendix

A. Procedure to create predecessor set of an entity.

Procedure: create-pred

Input:

x - Entity x (an event or a state)

Output:

$P_x = \{\text{Set of predecessors of } x\}$

Steps:

1. For each row r_k of transition table F where x is output
 - a. Get e_{in} and s_{in} of r_k with proper time relation
 - b. Set e_{in} and s_{in} as k^{th} predecessor p_k of entity x
2. Return P_x , set of all such p_k 's

B. Procedure to check existence of an entity in a scenario.

Procedure: check-existence

Input:

x - Entity x (an event or a state)

Output:

1. Yes (x exist in T_n), or
2. No (x does not exist in T_n), or
3. May-exist with $k+1$ choices.

Steps:

1. Find m , number of entities in T_n that are compatible with entity x
2. If ($m = 0$), return **No**
3. If ($m > 0$)
 - a. Do **test-exist** with each of these m entities
 - i. If output of **any** of the m tests is 'Old', return **Yes**.
 - ii. If outputs of **all** of the m tests are 'New', return **No**.
 - iii. Otherwise, let k is the number of entities for which **test-exist** outputs 'May-be-old'
 - iv. Return **May-exist** with $(k+1)$ choices
//Choice 1 represents that case when 'New' and the rest k choices represent each of
//the k cases that x is compatible with and potentially could be the same as x .

C. Procedure to enumerate all child nodes rooted at T_n .

Procedure: enumerate

Input:

T_n - A scenario described in a search tree node.

Output:

1. All child nodes rooted at T_n , or

Steps:

// x is an entity of T_n

// P_x is set of predecessors of x

// y is an unjustified entity of T_n

// Y is the set of unjustified entities of T_n

// Z_y is the set of predecessors of y

1. Set Y to NULL

2. For each entity x of the scenario T_n

- a. Use **create-pred** to get P_x

- b. For each predecessor p_k of P_x

- i. Use **check-existence** to find if p_k exist in T_n

- ii. If it returns 'Yes', skip x

- iii. If it returns 'No', add x to Y and add p_k to Z_y

- iv. If it returns 'May-exist'

1. Add x to Y

2. Add p_k to Z_y (choice 1)

3. Add each of k equations of test-exist tests as an element of Z_y

3. If ($Y = \text{NULL}$), return Y

// All entities of T_n are justified, no child is generated

4. For each entity y of Y

- a. Take cross product of Z_y 's

5. Return the result of cross product as child nodes of T_n

D. Test scenario generation procedure.

Procedure: create-tree

Input:

T_n - A scenario

Output:

O - Output scenarios leading to T_n

Steps:

1. Specify the scenario using implication using **specify-scenario** procedure

2. If it returns 'prune-branch', prune the branch. Else, continue to step 3

3. Enumerate child nodes using **enumerate**

- a. If **enumerate** returns NULL, add the output scenario to O

- b. Otherwise, for each child node that **enumerate** returns

- i. Use **create-tree** to generate scenario

E. Procedure to perform implication of a successful reception.

Procedure: succ-rx

Input:

$m - RE_{i,j}$ at t_v (an RE event), $m - RS_{i,j}$ at t_u (corresponding RS event) at node k , a scenario T_n

Output:

1. Consistent, or

2. Prune

Steps:

1. For all node p such that $k \in G_p$
 - a. For all node q such that $q \in G_p$ and $(p \ll q)$
 - i. For each RS or RE event e from event list of P
 1. Add event $e_{p,q}$ at t_w to PL_e of T_n
 2. Add equation $t_u \leq t_w \leq t_v$ to PL_{SOI} of T_n
 3. Check if $e_{p,q}$ at t_w exist in H_e of T_n using **check-exist**
 - a. If exist, return Prune
 2. Return Consistent.

F. Procedure to perform implication of a suppressive timer.

Procedure: supp-timer

Input:

$mT - TE_{i,i}$ at t_v (expiration of timer), $mT - TS_{i,i}$ at t_u (corresponding start), a scenario T_n

Output:

1. Consistent, or
2. Prune

Steps:

1. Get the event $e_{j,k}$ for which the timer is waiting
2. Add event $e_{j,k}$ at t_w to PL_e of T_n
3. Add equation $t_u \leq t_w \leq t_v$ to PL_{SOI} of T_n
4. Check if $e_{j,j}$ at t_w exist in H_e of T_n using **check-exist**
 - a. If exist, return Prune
2. Return Consistent.

G. Procedure to perform implication of a non-suppressive timer.

Procedure: non-supp-timer

Input:

$mT - TE_{i,i}$ at t_v (expiration of timer), $mT - TS_{i,i}$ at t_u (corresponding start), a scenario T_n

Output:

1. Consistent, or
2. Prune

Steps:

1. For all node j such that $j \in G_i$
 - a. For each TS or TE event e from event list of P
 - i. Add event $e_{i,j}$ at t_w to PL_e of T_n
 - ii. Add equation $t_u \leq t_w \leq t_v$ to PL_{SOI} of T_n
 - iii. Check if $e_{i,j}$ at t_w exist in H_e of T_n using **check-exist**
 1. If exist, return Prune
2. Return Consistent.

H. Procedure to check consistency of state history.

Procedure: state-cons

Input:

T_n

Output:

1. Consistent, or
2. Prune

Steps:

1. For all entries $i: \langle s_1, t_1, t_2 \rangle$ of state history H_s of scenario T_n
 - i. For all entries $j: \langle s_2, t_3, t_4 \rangle$ of state history H_s of scenario T_n and ($i = j$)
 1. Check if interval $[t_1, t_2]$ and $[t_3, t_4]$ overlap using **test-interval-overlap**
 - a. If overlap, return Prune
 2. Return Consistent.

I. Procedure to do unique backward implication.

Procedure: bk-implication

Input:

x - Entity x (an event $e_{i,j}$ at t_x or a state $i: \langle s_x, t_x, don'tcare \rangle$)

Output:

1. Consistent, or
2. Prune, or

Steps:

1. Find N , number of rows of transition table F where x is output
//Let R_N is the set of rows
2. If ($N=1$), let r_k is the row
 - a. Add $i: \langle s_{in-k}, unknown, t_x \rangle$ and $i: \langle s_{out-k}, t_x, unknown \rangle$ to H_s
 - b. return **process-event**($e_{in-k}att_x$)
3. If ($N>1$), go to step 4
4. set $R_K = \mathbf{elim-rows}(x, R_N)$
// Let K is the cardinality of set R_K
5. If ($K=1$) go to step 6. Else, go to step 7.
//Let r_j is the only row in R_K
6. Find from H_s , if $i: \langle s_{in-j}, t_u, t_v \rangle$ exist using **check-exist**
 - a. If exist,
 - i. Use **find-relation**(t_u, t_x) and **find-relation**(t_v, t_x)
 - ii. If ($t_u < t_x$) and ($t_v \geq t_x$)
 1. Add $i: \langle s_{in-k}, unknown, t_x \rangle$ and $i: \langle s_{out-k}, t_x, unknown \rangle$ to H_s
 2. return **process-event**($e_{in-k}att_x$)
 - iii. Else, return Consistent
 - b. Else, return Consistent
7. Set $y = \mathbf{row-intersection}$
8. If ($y \neq \text{NULL}$)
 - a. Add the state in H_s if y is a state
 - b. return **process-event**(y) if y is an event
9. Else, set total-exist = 0
10. For each row r_j of set R_K
 - a. Find from H_s , if $i: \langle s_{in-j}, t_u, t_v \rangle$ exist using **check-exist**
 - i. If exist,
 1. Use **find-relation**(t_u, t_x) and **find-relation**(t_v, t_x)
 2. If ($t_u < t_x$) and ($t_v \geq t_x$), total-exist++
11. If (total-exist > 1), return Prune
12. Else, return Consistent

J. Procedure to eliminate predecessors.

Procedure: elim-rows

Input:

1. x - Entity x (an event $e_{i,j}$ at t_x or a state $i: \langle s_x, t_x, don'tcare \rangle$)

2. R_N = a set of N numbers (row numbers)

Output:

- R_K = a set of K numbers (row numbers)

Steps:

1. Set R_J and R_K to empty
2. For each row r_j of R_N
 - a. Find from PL_s , if $i: \langle s_{in-j}, t_u, t_v \rangle$ exist using **check-exist**
 - i. If exist, use **find-relation**(t_u, t_x) and **find-relation**(t_v, t_x)
 1. If ($t_u < t_x$) and ($t_v \geq t_x$)
 2. Else, add r_j to R_J
3. For each row r_j of R_J
 - a. Find from PL_s , if $i: \langle s_{in-j}, t_u, t_v \rangle$ exist using **check-exist**
 - i. If exist, use **find-relation**(t_u, t_x) and **find-relation**(t_v, t_x)
 1. If ($t_u < t_x$) and ($t_v \geq t_x$), add r_j to R_K
4. Return R_K

K. Procedure to find common items of a given set of rows.

Procedure: row-intersection

Input:

2. R_K = a set of K numbers (row numbers)

Output:

1. an entity: an event or a state, or
2. NULL

Steps:

1. For each row r_2 to r_K of R_K
 - a. If $s_{in-1} \langle \rangle s_{in-j}$ go to step 3
2. Return s_{in-1}
3. For each row r_2 to r_K of R_K
 - a. If $e_{in-1} \langle \rangle e_{in-j}$ return NULL
4. Return e_{in-1}

L. Procedure to find common items of a given set of rows.

Procedure: process-event

Input:

2. $e_{i,j}$ at t_u

Output:

1. Consistent, or
2. Prune

Steps:

1. Find if the event exist in PL_e using **check-exist**
2. If exist,
 - a. Return Prune
3. Find type of the event.
 - a. If RE event
 - i. Use **proc-succ-rx**
 - ii. If **proc-succ-rx** returns Prune, return Prune
 - iii. Else, continue
 - b. If suppressive timer expiration event
 - i. Use **proc-supp-timer**

- ii. If **proc-supp-timer** returns Prune, return Prune
 - iii. Else, continue
 - c. If non-suppressive event
 - i. Use **proc-non-supp-timer**
 - ii. If **proc-non-supp-timer** returns Prune, return Prune
 - iii. Else, continue
- 4. Find if the event exist in H_e using **check-exist**
- 5. If does not exist,
 - a. Add the event in H_e
- 6. Return Consistent

M. Procedure to specify a scenario.

Procedure: specify-scenario

Input:

T_n - A scenario described in a search tree node.

Output:

1. Consistent, or
2. Prune

Steps:

1. While new information is generated into T_n
 - a. For each entity x of T_n
 - i. $bk = \mathbf{bk-implication}(x)$
 - ii. $fw = \mathbf{fw-implication}(x)$
 - iii. If ($bk = \text{Prune}$) OR ($fw = \text{Prune}$)
 1. Return Prune
 - iv) Else, continue
2. Return Consistent

References

- [1] G. Holzmann, "Design and Validation of Computer Protocols", Prentice Hall, ISBN 0L35399254.
- [2] F. Lin, P. Chu, and M. Liu, "Protocol Verification Using Reachability Analysis", Computer Communication Review, vol. 17, no. 5, pp. 126-135, October 1987.
- [3] R. Alur and D.L. Dill, "A theory of timed automata", Theoretical Computer Science 126:183-235, 1994.
- [4] M. Kwiatkowska, G. Norman and D. Parker, "PRISM 2.0: A Tool for Probabilistic Model Checking", 1st International Conference on Quantitative Evaluation of Systems (QEST'04), pp. 322-323, IEEE Computer Society Press. September 2004.
- [5] M. Kwiatkowska, G. Norman and D. Parker, "Probabilistic model checking in practice: Case studies with PRISM", Special issue of ACM Performance Evaluation Review on Performance and Verification, 32(4), pp. 16-21, March 2005.
- [6] M. Kwiatkowska, G. Norman and J. Sproston. "Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol", PAPM/PROBMIV '02, volume 2399 of LNCS, pp. 169-187, July 2002.
- [7] M. Dufлот, M. Kwiatkowska, G. Norman and D. Parker, "A Formal Analysis of Bluetooth Device Discovery", 1st International Symposium on Leveraging Applications of Formal Methods (ISOLA'04), November 2004.

- [8] A. Helmy and D. Estrin, "Simulation based 'STRESS' Testing Case Study: A Multicast Routing Protocol", 6th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), July 1998, Montreal, Canada.
- [9] A. Helmy, D. Estrin, and S. Gupta. "Fault-oriented Test Generation for Multicast Routing Protocol Design", Joint International Conference on Formal Description Technique/Protocol Specification, Testing, and Verification (FORTE/PSTV), pp. 93-109, November 1998.
- [10] R. Jurdak, C. Lopes, and P. Baldi, "A Survey, Classification and Comparative Analysis of Medium Access Control Protocols for Ad Hoc Networks", IEEE Communication Surveys and Tutorials, 1st quarter 2004, vol. 6, no. 1, pp. 2-16.
- [11] IEEE Std 802.11-1997 Information Technology - telecommunications And Information exchange Between Systems-Local And Metropolitan Area Networks-specific Requirements-part 11: Wireless Lan Medium Access Control (MAC) And Physical Layer (PHY) Specifications IEEE Std 802.11-1997,Page(s): i -445
- [12] P. Karn, "MACA - A New Channel Access Method for Packet Radio", 9th ARRL/CRRL Computer Networking Conference, pp. 134-140, 1990.
- [13] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: A Media Access Protocol for Wireless LANs", ACM Special Interest Groups on Data Communication (SIGCOMM), pp. 212-225, September 1994.
- [14] C. Fullmer and J. Garcia-Luna-Aceves, "Floor Acquisition Multiple Access (FAMA) for Packet-Radio Networks", Conference on Applications, Technologies, Architectures and Protocols for Computer Communication, pp. 262-273, 1995.
- [15] F. Iulucci, M. Gerla, and L. Fratta, "MACA-BI (MACA BY Invitation): A Receiver-Oriented Access Protocol for Wireless Multihop Networks", 8th IEEE International Symposium on Personal, Indoor and Mobile Radio Communication, vol 2. 1997, pp 435-439.
- [16] L. Bononi, M. Conti, and L. Donatiello, "Distributed Contention Control Mechanism for Power Saving in Random Access Ad-Hoc Wireless Local Area Networks", International Workshop on Mobile Multimedia Communication, IEEE, August 1999, pp. 114-123.
- [17] J. Garcia-Luna-Aceves and A. Izamaloukas, "Reversing the Collision Avoidance Handshake in Wireless Networks", ACM/IEEE International Conference on Mobile Computing and Networking, 1999, Seattle, Washington.
- [18] S. Khurana, A. Kahol, A. Jayasumana, "Effect of Hidden Terminals on the Performance of IEEE 802.11 MAC Protocol", IEEE 23rd Annual Conference on Local Computer Networks (LCN), pp 12-20, October 1998, Boston, Massachusetts.
- [19] S. Khurana, A. Kahol, S. Gupta, P. Srimani, "Performance evaluation of distributed co-ordination function for IEEE 802.11 wireless LAN protocol in presence of mobile and hidden terminals", 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), March 1999, College Park, Maryland.
- [20] C. Koksal, H. Kassab, H. Balakrishnan, "An Analysis of Short-Term Fairness if Wireless Media Access Protocols", ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pp 118-119, 2000.
- [21] S. Sharma, "Analysis of 802.11b MAC: A QoS, Fairness, and Performance Perspective", Technical Report TR-126, Department of Computer Science, State University of New York, Stony Brook, January 2003.
- [22] L. Bononi, M. Conti, E. Gregori, "Runtime Optimization of IEEE 802.11 Wireless LANs Performance", IEEE Trans. Parallel Distrib. Syst, pp. 66-80, January 2004.
- [23] S. Begum, S. Gupta, A. Helmy, "Test Generation Framework for Performance Evaluation of Wireless AdHoc MAC Protocols", Technical report, <http://www-scf.usc.edu/sbegum/tech-report-eotg-1.pdf>.

- [24] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, "Advances in Network Simulation", IEEE Computer, 33 (5), pp. 59-67, May 2000.
- [25] LINDO systems, <http://www.lindo.com>.