Design and Development of Intrusion Detection System: A Discrete Event System Approach

Thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

By

Ferdous Ahmed Barbhuiya

Under the guidance of

Dr. Sukumar Nandi &

Dr. Santosh Biswas

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati 781039, India

April 2014

Declaration

I hereby declare that,

1. The work contained in this thesis is original and has been done by myself under the supervision of my advisors.
2. To the best of my knowledge this work has not been submitted elsewhere in any university for the award of any degree or diploma.
3. I have given appropriate credit to the original authors whenever I used materials (data, text or figures) from other sources.
4. Whenever I have quoted written material from other sources, I have put them under quotation marks and given due credit to the sources by citing the sources and giving required details in the references.
5. I have followed the norms and guidelines given in Ethical Code of Conduct of the Institute.

Place: IIT Guwahati

Date: **(Ferdous Ahmed Barbhuiya)**

Certificate

This is to certify that this thesis entitled **"Design and Development of Intrusion Detection System: A Discrete Event System Approach"** *submitted by Ferdous Ahmed Barbhuiya, to the Indian Institute of Technology, Guwahati, for partial fulfillment of the award of the degree of Doctor of Philosophy, is a record of bona fide research work carried out by him under our supervision and guidance.*

The thesis, in our opinion, is worthy of consideration for award of the degree of Doctor of Philosophy in accordance with the regulations of the institute. To the best of our knowledge, the results embodied in the thesis have not been submitted to any other university or institute for the award of any other degree or diploma.

Place : I.I.T. Guwahati, India
Date:

(Sukumar Nandi)
Dept. of Computer Science and Engineering,
Indian Institute of Technology, Guwahati.

Place: I.I.T. Guwahati, India
Date:

(Sanstosh Biswas)
Dept. of Computer Science and Engineering,
Indian Institute of Technology, Guwahati.

Acknowledgement

At the outset, I express my sincere gratitude and thanks to my supervisors Prof. Sukumar Nandi and Prof. Santosh Biswas. It has been a honour to be their PhD student. I appreciate all their contributions of time, effort and ideas to make my Ph.D. experience productive and enjoyable. The joy and enthusiasm for research was contagious and motivational for me, even during tough times in the Ph.D. pursuit. I am also thankful for the excellent example they have provided as a successful scientists and professors.

I am grateful to all the doctoral committee members for their valuable feedback on my ideas during my research. I am thankful to all faculty members and staff of the department for their kind cooperation and encouragement. I am indebted to many people who helped me and supported me in bringing forth this thesis.

A special word of thanks goes to Department of Information Technology (DIT), New Delhi for sponsoring a security project to IIT Guwahati in which I was a project staff during the course of my PhD. The project facilitated a wonderful security laboratory without which many of the implementation results of my work would have been a mere dream.

It was a great fun and source of ideas and energy to have friends like Neminath, Roopa, Ritesh, Vaibhav, Maheshweta, Prithu, Gunjan, Niteesh and sanketh to name a few, with whom I closely worked. Leaving apart technical work, I enjoyed company of Bidyut, Amrita, Sandip, Maushumi, Suchetana and Dhruwajitha and I thank all of them.

Last but most important, I would like to thank my family for all their love and encouragement. For my parents who raised me with a love of science and supported me in all my pursuits. I am grateful to all my family members for their support. And most of all for my loving, supportive, encouraging, and patient wife Nazia whose faithful support during the final stages of this Ph.D. is so appreciated. At the end, my little angel Fariha, who is such a nice kid that I could comfortably write my thesis even though she is a newborn. Thank you.

Place: IIT, Guwahati

Date: **(Ferdous Ahmed Barbhuiya)**

Abstract

With the rapid increase of security threats in Internet, Intrusion Detection System(IDS), a hardware or software that monitors network or host activities for malicious behavior, is an indispensable component of Network Security. Among the two prevalent IDS designing techniques, signature based IDSs can detect known attacks only while anomaly based systems can detect both known and unknown attacks, but generates large number of false alarms. There are classes of attacks like ARP based attack, ICMP based attack, TCP low rate DoS attack etc. which escape detection by both signature and anomaly IDSs. This thesis proposes a Discrete Event System(DES) based approach to design IDS for attacks across different network layers. DES models are designed for the system under normal and failure conditions where attacks are mapped to failures. A state estimator called diagnoser is designed which observes sequences of events generated by the system to decide whether the states through which the system traverses correspond to the normal or faulty DES model. The diagnoser acts as the IDS engine. For detecting ARP based attacks, an active probing mechanism based on ARP requests and responses is used. Active DES framework is adopted to model ARP based attacks using a controllable event (ARP probe) which creates difference in sequence of events for normal or attack condition. Next, to handle network uncertainties due to presence of congestion, for detecting ICMP based attack, I-diagnosis framework of DES has been adopted where diagnosis is tested only in those sequence of states where a fault is followed by a indicator event. Redundant states of diagnoser of *I*-diagnosis framework are removed and a reduced detector is also proposed to improve complexity. Further, in Induced Low Rate TCP DoS attack, the attack and genuine sequence of state differs with some probability. So to detect this attack, Stochastic DES framework has been adapted where attack case can be identified with some probability. Lastly, considering the migration from IPv4 to IPv6 addressing in the Internet, detection mechanism for NDP based attacks of IPv6 network is proposed. To tackle the challenge of presence of error in building complex DES model manually for NDP related attacks in IPv6, LTL based DES framework is adopted. All proposed detection mechanisms are implemented in testbed and the results show the effectiveness of the systems in terms of accuracy and detection rate.

List of abbreviations

ARP	Address Resolution Protocol
CAM	Content Addressable Memory
CGA	Cryptographically Generated Addresses
CVE	Common Vulnerability Exposure
DAD	Duplicate Address Detection
DES	Discrete Event System
DNS	Domain Name Server
DoS	Denial of Service
FDD	Failure Detection and Diagnosis
FN	False Negative
FP	False Positive
FSA	Finite State Automata FSM Finite State Machine
HIDS	Host based Intrusion Detection System
HTTP	Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol
ICMPv6	Internet Control Message Protocol version 6
IDS	Intrusion Detection System
IKE	Internet Key Exchange
IP	Internet Protocol
IPv6	Internet Protocol version 6
IPS	Intrusion Prevention System
LAN	Local Area Network
LTL	Linear-time Temporal Logic
MAC	Media Access Control
MiTM	Man-in-The-Middle
MSS	Maximum Segment Size
NA	Neighbor Advertisement
NDP	Neighbor Discovery Protocol
NS	Neighbor Solicitation
NIC	Network Interface Card

NIDS	Network based Intrusion Detection System
NUD	Network Unreachability Detection
NVD	National Vulnerability Database
PKI	Public Key Infrastructure
RA	Router Solicitation
RFC	Request for Comments
RS	Router Advertisement
RTO	Retransmission Time out
SEND	Secure Neighbor Discovery
TCP	Transmission Control Protocol
TP	True Positive
UDP	User Datagram Protocol

Contents

List of Figures

List of Tables

Chapter 1

Overview

Heavy reliance on the Internet and worldwide connectivity has greatly increased the potential damage that can be inflicted by attacks launched over the Internet against remote systems. It is difficult to prevent such attacks by the use of security policies, firewall, or other mechanisms because system and application software always contain unknown weaknesses or bugs. In addition, complex, often unforeseen, interactions between software components and/or network protocols are continually exploited by attackers. Successful attacks inevitably occur despite the best security precautions. One reason for this is the explosive growth of the Internet and the large number of networked systems that exist in all types of organizations. The increased number of networked machines has led to a rise in unauthorized activity [1], not only from external attackers, but also from internal sources such as disgruntled employees and people abusing their privileges for personal gain [2]. Moreover due to lack of proper authentication of many communicating entities in the TCP/IP layering architecture, each layer is prone to various threats and vulnerabilities. This mandates the requirement for a suitable detection system mechanism to monitor the possible attacks in a layered manner.

There are number of security components employed in a network by the security administrators like firewall, proxy server, access control list, Intrusion Prevention System (IPS) and Intrusion Detection System (IDS). IDS have become an essential component of computer security to detect attacks before they inflict widespread damage and the field of intrusion detection has received increasing attention in recent years.

Intrusion is any set of actions that attempt to compromise confidentiality, integrity or availability of a computer resource [3]. An IDS is a device or software application that monitors network and/or system activities for malicious behavior or policy violations and prepares a report for system administrator. IDS mainly involves detecting whether some entity has attempted or worse gained access to system resources in a nondiplomatic way.

An IDS can be host based or network based depending on whether it monitors a single host or a network. A Host based IDS (HIDS) is an IDS that monitors and analyzes the internals of a computing system like, execution of applications, system calls etc. to detect malicious behavior. A Network based IDS (NIDS) is an IDS that detects malicious behavior by monitoring network traffic. Rest of this thesis mainly deals with NIDS and henceforth unless otherwise mentioned, the term IDS refers to NIDS.

Rest of the chapter is organized as follows. Section 1.1 presents a taxonomy of IDSs based on types of attacks they can detect. A brief report on major shortcoming of different classes of IDSs and issues with such techniques are discussed in Section 1.2. This section also develops motivation for the work done in the thesis and highlights the major contributions. In Section 1.3 the organization of the thesis in six chapters is discussed.

1.1 IDS taxonomy based on attack detection

IDSs can be classified into two classes based on the type of attacks they detect.

- Signature based IDS.
- Anomaly based IDS.

1.1.1 Signature based IDSs

An IDS which uses signatures for detecting attacks is called signature based IDS. A signature is a pattern that is looked for, in network traffic by IDS to detect attacks. A set of known attack signatures are stored in a database and the network or system activity is compared with these signatures. If a match is found an alarm (used interchangeably with "alert" in this thesis) is raised. Number of signature detection systems have been proposed in the literature. Snort [4] and EMERALD [5] are some of the prominently used signature detection systems.

A simple example of a Snort signature for "*land*" attack is as follows.
alert ip any any − > any any (*msg* : *"BAD TRAFFIC sameSRC/DST"*; *sameip*; *reference* : *cve*, *CVE−1999−0016*; *reference* : *url*, *www.cert.org/advisories/CA−1997−28.html*; *classtype* : *bad − unknown*; *sid* : 527; *rev* : 3;)
This signature filters out network packet having same source and destination IP address and raises an alarm.

1.1.2 Anomaly based IDSs

Signature based IDS can detect only attacks whose syntax or behavior is known. However if an attack is new or its syntax and behavior is unknown, normally another class of IDS termed as anomaly based IDS is used. Anomaly based IDS can detect both known and unknown attacks. It models benign behavior of a system with a profile and any deviation from the known profile is considered as intrusion. Benign profile is represented as a set of parameters (which are also called features) and are extracted from network packets. For example, number of TCP SYN packets observed within a time window is a feature. A review of various anomaly based IDSs can be found in [6, 7].

1.2 Limitations of existing IDS, Motivation and contribution of the Thesis

While Signature based IDS is effective at detecting attacks without generating an overwhelming number of false alarms, but it can detect only known attacks. Therefore such IDS must be constantly updated with signatures of new attacks. Many signature detectors are designed to use tightly defined signatures that prevent them from detecting variants of common attacks.

Anomaly IDS can detect unusual behavior and thus have the ability to detect symptoms of attacks without specific knowledge of details and can produce information that can in turn be used to define signatures for misuse detectors. However, it produces a large number of false alarms due to the unpredictable behaviors of users and networks. Further, it requires extensive training sets of system or network event records in order to characterize normal behavior patterns, which is a difficult task.

There are certain class of attacks which cannot be detected with either of these two types of IDSs. Some of them are ”Address Resolution Protocol (ARP) based Insider Attack”, ”Attack through Internet Control Message Protocol (ICMP) messages”, ”Attack on Transmission Control Protocol (TCP)” etc. which are detailed next.

1.2.1 ARP based Attacks

At the data link layer computers use another address known as Media Access Control (MAC) address or hardware address. To deliver a packet to correct machine the Internet Protocol (IP) address has to be mapped to some MAC address. ARP is responsible for finding the MAC address given the IP address. Data link layer use the MAC address of the destination machine for sending the packets. If the host sending the packets does not know the MAC address of the destination host, it sends a broadcast request asking “What is the MAC address corresponding to the IP address”. The host which has the IP address in the broadcast message sends a unicast reply message to the sender mentioning its MAC address. In order to reduce the number of broadcast requests each machine maintain a table termed as ARP cache which holds the mapping between the IP and MAC. The entries in the ARP cache can be either static or dynamic. In the dynamic cache the entries are erased as they get older than a predefined duration. The problem with ARP is that, it is a stateless protocol. Any host after receiving any ARP response message will update its cache without verifying wether it has earlier sent a request corresponding to the response. This statelessness is the major security loophole in the protocol and enables malicious hosts to craft custom ARP packets with forged IP-MAC pairs, which leads to ARP spoofing attacks.

The attack is not detectable by signature based IDSs as there is no difference between normal and spoofed ARP packets. To perform this attack, an attacker needs to send very few ARP packets, typically one or two leaving all statistical properties of network traffic intact. So, no anomaly is noticed in the network when such attack occurs.

There are number of passive solutions proposed in the literature to detect, mitigate and prevent such attacks. The most foolproof way to prevent ARP attacks is to manually assign fixed IP to all hosts and maintain the static IP-MAC pairings at all the hosts [8]. However, in a dynamic environment this is not a practical solution. Several hardware

(switch) [9] and software [10, 11, 12] based solutions have also been proposed which are flexible than static IP-MAC pairings. These schemes involve memorizing the IP-MAC pairs first obtained from network traffic and take appropriate actions (like blocking of IP) on passive observation of any change thereafter. The problem with these passive observation based approaches are, if the first received packet itself is having a spoofed MAC address then the whole scheme fails. Further, any genuine change in IP-MAC pair will be discarded (e.g., when notified by Gratuitous request and reply). Signature based IDSs have also been applied to detect ARP based attacks [13]. The main problem with signature based IDSs is that of a high number of false alarms [13]. Reported results show that false negative rates are as high as 40%.

Active techniques for detecting ARP attacks have also been proposed [14, 15] where the IDS actively sends probe packets (active probes) to hosts in addition to observations (like changes of IP-MAC pairs). In [14], a database of known IP-MAC pairs is maintained and on detection of a change the new pair is actively verified by sending a probe with TCP SYN packet to the IP under question. The genuine host will respond with a SYN/ACK or RST depending on whether the corresponding port is open or closed. While this scheme can validate the genuineness of IP-MAC pairs, it violates the network layering architecture. An active scheme for detecting Man-in-The-Middle(MiTM) attack is proposed in [15]. First, all hosts with IP forwarding are detected. Following that, the IDS attacks all such hosts one at a time and poison their caches in a way such that all traffic being forwarded by the attacker reaches the IDS (instead of the host, the attacker with IP forwarding wants to send). So, the IDS can differentiate the real MiTM attackers from all hosts with IP forwarding.

Although active techniques for ARP attack detection are superior compared to passive ones, however, involves extra traffic due to probing. Experiential results in [15] have shown that the overhead of extra traffic is negligible given the bandwidth of modern Local Area Networks(LAN).

From the review, it may be stated that an ARP attack detection/preventation scheme needs to have the following features

- Should not modify the standard ARP or violate network layering architecture
- Should generate minimal extra traffic in the network
- Should not require patching of all systems.

Failure Detection and Diagnosis (FDD) of Discrete Event System (DES) has been adapted for IDS in [16, 17, 18]. A DES framework is characterized by a discrete state space and event driven dynamics. DES paradigm has been widely used for Failure Detection and Diagnosis (FDD) because of modeling simplicity and the associated algorithms [19]. Further, for most of the systems can be modeled as DES using some level of abstraction. The basic idea is to develop a DES model for the system under normal condition and also under each of the failure conditions. Following that, a state estimator called diagnoser (or detector, if only detection of failure is required) is designed which observes sequence of events generated by the system to decide whether the states through which the system traverses correspond to the normal or faulty DES model. DES based IDSs [16, 17, 18] basically map attacks to failures and the diagnoser is implemented as the IDS engine. The major motivation of using DES framework for IDS is that it enables designing the IDS without changing the protocols involved. Broadly speaking, the major areas of application of FDD of DESs are physical systems like nuclear reaction chambers, chemical plants etc. whose designs are not changed for failure detection. So DES diagnoser can work without changing the system design.

Neminath et al. in [17, 18] have used DES for detecting ARP attacks and have illustrated that the concerned IDS use passive DES framework [20]. The probes are assumed to be sent by some external supervisory controller which is not present in the model. However, as already discussed, for ARP attacks active schemes [14, 15] outperform the passive ones [8, 11, 12]. The active modeling of attack detection mechanism for ARP based attack detection becomes necessary so that there is no external component in the model to facilitate validation. The proposed IDS [17, 18] assumes that there is some external system outside IDS which sends the probe and the IDS passively monitors the response and the framework itself is not active. In this contribution we propose an IDS for ARP attacks based on FDD theory of active DES framework. A supervisory controller (of IDS) receives all traffic due to port mirroring and sends probe requests to verify genuineness of ARP requests and responses; these correspond to controllable events. The diagnoser passively monitors all events namely, ARP requests, APR responses, probe responses etc. to determine whether it is attack or normal situation.

The contributions of the proposed detection system are enumerated below:

1. An active IDS for ARP attacks has been developed by adapting the FDD theory of

active DES framework [21]. As DES paradigm is state transition system so it suffers from state explosion problem. Classical systems handled by DESs typically have states in the range of thousands, which can be handled. However, in case of network protocols the range of the parameters are very large e.g., the size of domain of IP address can be $O(256.256.256.256)$. So while adapting the active DES paradigm [21] for developing IDS, state explosion problem is addressed by use of model variables.

2. The active DES based IDS has the following salient features:

   (a) Follows standard ARP and do not violate the principles of network layering structure.

   (b) Generates minimal extra traffic in the network, as active probing is done only when an unverified IP-MAC pair is seen in the traffic. This is achieved by maintaining tables for authenticated and spoofed IP-MAC pairs.

   (c) Being a network based IDS, it involves installation in just one host in the network.

   (d) As this is a software approach, it does not require installation of any extra hardware. The only requirement is that the switch should have port mirroring facility.

   (e) High detection rate and accuracy is achieved at reasonable resource overhead in terms of bandwidth, CPU utilization and memory.

1.2.2 Attack through ICMP messages

ICMP is used in network layer to send one-way informational messages to implement various error-reporting, feedback and testing capabilities. ICMP is used in the Internet to perform the fault-isolation function, which is the group of actions that hosts and routers take to determine if there is some network failure. There is no authentication option available in ICMP, which leads to attacks using ICMP. This can result in a Denial of Service (DoS), or allowing the attacker to intercept packets.

ICMP messages can be classified into two categories depending upon their operation: *Error messages* and *Informational messages*. Error messages are used to report the different kind of errors generated in the delivery of packets. Informational messages are used for diagnostics, testing and other informational purposes.

Examples of ICMP *Informational messages* based attacks are as follows.

DoS attacks primarily use either the ICMP "Echo request/reply" or "Router Advertisement" messages. An attacker can forge one of these ICMP messages, and send it to one or both of the communicating hosts to disrupt their connection. ICMP messages can also be used to intercept packets by using the ICMP "Redirect" message which is commonly used by gateways when a host has mistakenly assumed the destination is not on the local network. By forging an ICMP "Redirect" message, attacker can siphon traffic from a host by advertising its own IP address as the new router in the ICMP "Redirect" message.

Some examples of ICMP *Error messages* based attacks are as follows.

Destination Unreachable messages [22] are used to inform the host that destination is unreachable due to some reason. These errors may be generated as a result of TCP, UDP or another ICMP transmission. The messages may be generated either from gateway or from host. The various attacks related to these messages are host, port, protocol unreachable etc. These error messages can be spoofed by the attacker and can be used to stop a host connecting to the destination network which results in DoS. Host Unreachable error, one kind of destination unreachable error message is generated by the boundary gateway or router of different subnet. It is send by router when it receives a datagram which it cannot deliver or forward to destination host. An attacker can spoof this error message and stop the source from communicating with the destination host.

Different detection mechanisms are available for identifying the attacks related to each of these ICMP messages. They are as follows.

In Signature-based detection systems, based on rulesets, ICMP packets are dropped when the signatures are matched. Cisco Firewall [23] has a set of signatures to protect the internal network. For instance, it checks if ICMP packets have "More Fragments" flag set to 1 and whether the length in IP header is greater than 1024 bytes. Likewise, another signature for protection against DoS in Cisco routers limits the number of ICMP unreachable packets to one per 500 ms. The whitepaper [24] suggests to block/drop the packets related to ICMP redirect and ICMP unreachable messages to avoid DoS attacks. As this kind of detection systems have predefined rules or signatures for a particular attack, even slight variation in attacks would not be detected by these IDSs. In [25], Chau suggests the router configuration to block all the outbound packets that indicate a source address which is not a part of the subnet. These router configurations block the essential ICMP

packets which is not desirable. Every ICMP message has its own significance and hence they should not be dropped blindly. In [26], ICMP traceback messages are used to learn the path that packets take through the Internet. With enough traceback messages from routers along the path traffic, source and path of forged packets can be determined. But such extensions to ICMP messages require changes to the global routing infrastructure which is not practical. In [27], secure mechanism for router discovery for hosts in IPv6 is presented. Cryptographic techniques to add authenticity to the ICMP messages may be used, but these have costly computational overhead and high storage resource requirement.

For these attacks, writing a signature is not possible as the syntax and sequence of network traffic under normal and compromised situations does not differ. So, it is not possible to detect those attacks using signature based IDS. Further these attacks do not lead to any significant deviation in network traffic characteristics. So anomaly based IDS also can not detect these attacks effectively. Again for these attacks generate extremely high alarms. These drawbacks in the existing techniques motivated us to propose a novel approach for detecting ICMP based attacks.

For ARP based attacks, the system is found to be diagnosable using active DES framework. This means if an attack takes place, it is possible to conform its occurrence by analyzing packet sequence, within a finite amount of time, using a controllable event (active event called probe). But ICMP attacks are detectable under certain network conditions and undetectable in other network conditions. In some cases it is not possible to deduce that an attack has occurred within finite delay after the attack takes place. For example, if we try to ascertain the genuinity of an ICMP error message (say destination unreachable message) by using an active probe, the resulting reply of the probe might not come back if there is congestion in the network. Thus the detector of these attacks does not always end in a normal/attack certain state rendering it to be nondiagnosable. Being congestion is an uncontrollable event, active DES framework can not be applied for ICMP based attack.

In [20] authors have proposed a framework called I-diagnosability, for partial diagnosis problems. Here some indicator events are defined and failure diagnosis is tested only in those paths where a fault is followed by an indicator event. The paths where there are no indicator events, may contain some uncertain states that does hinder diagnosis of the system as a whole. For ICMP related attacks, the paths where there is issue like congestion, no indicator event is available. Thus this restricted diagnosability framework

[20] is adopted for modeling and designing IDS for ICMP related attacks.

The salient contribution of the scheme is enumerated below

1. An IDS for IMCP attacks has been developed by adapting the FDD theory of I-Diagnosability [20]. The following are the modifications in the I-Diagnosability which made is suitable for designing the IDS:

   (a) As in the case of active DES framework, I-Diagnosability also has the issue of state explosion which is addressed by augmenting the framework with model variables.

   (b) The *I*-detector used in I-Diagnosability framework [20], contains many redundant states. So a reduced *I*-detector (called *RI*-detector) has been developed by eliminating the redundant states and it has been shown that fault detection capability is not compromised.

2. The DES based IDS for ICMP has similar salient features as in the case for ARP namely, non requirement of change in ICMP, minimal extra traffic overhead, reasonable resource overhead, non requirement of patching individual hosts etc.

1.2.3 TCP Low rate DoS attack

TCP is a transport layer protocol which provides process to process reliable byte stream delivery. In addition it provides flow control, congestion avoidance and error control. Nearly all the application protocol that want process to process reliable delivery use this transport layer protocol. TCP is a complex and robust protocol composed of many algorithms which serves its purpose to great extent. As the main function of TCP is to provide reliable byte stream process to process communication little or almost no consideration was given while designing this protocol to the fact that algorithms used in TCP can be exploited by attackers. Attackers have exploited algorithm of TCP to launch various attacks like SYN flood attack, TCP session Hijacking, RST and FIN attack and attacks exploiting congestion avoidance algorithm. Low rate TCP-targeted attack and Induced low rate TCP-targeted attacks are such attacks that exploit congestion avoidance algorithm.

Low Rate TCP-targeted attack also called as shrew attack [28] was identified by Aleksandar Kuzmanovic and Edward W. Knightly in 2003. In this attack, an attacker exploits the congestion avoidance algorithm of TCP protocol. Attacker attacks by congesting the network for short duration; and subsequently keeps quiet for relatively longer duration. When it congest the network to the target server, other genuine TCP flows destined to that server loose all or few of their segments. This causes genuine TCP flows to slow down their transmission rate. According to congestion avoidance algorithm of TCP whenever a segment is lost and its acknowledgement does not come before Retransmission Time out (RTO) period, the rate of transmission by sender is reduced drastically to slow start phase with congestion window of 1 Maximum Segment Size(MSS). As the name suggest, while performing this attacks, the attacker keeps the average attack traffic rate low, so that it can not be identified by queuing techniques like RED-PD [29].

Many detection and protection schemes have been proposed against Low rate TCP targeted attacks. In [30] a fair queueing and randomization of RTO based solution is proposed which give equal chances to packet of each flow in the queue of router. But Fair Queue scheduling is not scalable. TCP implementation of every host needs to be changed to randomize RTO and also it can cause degradation of performance during non attack phase. Haibin Sun et al. in [31] proposed a solution in which Edge Router monitors traffic flow going out from it towards target server and uses resource restriction on suspected traffic. This may damage legitimate TCP flows as there is no distinct pattern available for this attack. Shevatker et al. in [32] proposed a solution which notes the arrival time of each packet and observes the time difference of packets for each flow to find the characteristics of shrew attack. Deficiency of this solution is that it has to maintain information for all TCP flows, which requires more processing and memory resources in each router. Also it cannot detect attack when it is executed in distributed fashion. Kwok et al. [33] proposed a scheme in which two time scales were maintained, one for finding burst length and one for attack period. Flows are monitored to see if traffic is coming at higher rate. This method cannot detect the attack when it is carried out in distributed manner as burst length or attack period for an individual attacker is not available. Luo et al. [34] proposed a scheme in which they studied incoming and outgoing TCP traffic to detect attack. However, it is hard to find optimal set of parameters that are sensitive enough to detect distributed low rate TCP-targeted attack while keeping false positive rate low. Chia-Wei Chang et al. [35]

proposed a solution based on modification in dropping technique of packets in the queue of router by classifying TCP flows according to ports. This solution can lessen the intensity of attack but cannot detect and prevent the attack. Also it is less effective if attacker's flow is of same class as of most of TCP flows destined to that server.

From the shortcoming of the existing approaches it is apparent that we need a scheme in which the solution does not require any TCP packet format modification, does not require any customization of any intermediary router or client host and does not require much resources.

The DES frameworks discussed till now is called state based framework because, the normal and faulty behavior of the system, also called specification, is modeled by a state-transition system. In ARP related attack, normal and attack sequence of events could be separated using active probes. ICMP attacks could be diagnosed by separating normal and attack event sequence in certain paths using I-diagnosis framework of DES. But in case of Low Rate TCP DoS attack, separation of normal and attack events are neither possible by active diagnosis nor by I-diagnosis framework because the attack and genuine traces may not clearly differ, but may differ with some probability.

For detecting this type of attack, we have adapted Stochastic DES framework [36]. State based DES is characterized by a discrete state space and event driven dynamics [19], but in stochastic DES, probabilities of occurrence of events are also specified. The basic idea is to develop a stochastic DES model for the system under normal condition and under each of the faulty (or attack) conditions. Following that, a stochastic state estimator called stochastic diagnoser is designed, which observes sequence of events generated by the system to decide whether the states through which the system traverses correspond to the normal DES model or faulty DES model and if the probability of the former decreases with time, attack is considered detectable.

The contributions of this scheme are enumerated below

1. A Stochastic DES framework based IDS has been designed to deal with Low Rate TCP-targeted DoS attack. To deal with very large size of domains of variables, in the adapted DES model, state variables are incorporated.

2. The Stochastic DES framework based IDS has minimal overhead in normal case and improves performance in attack situation by nullifying the effect of the attack.

1.2.4 Attack based on NDP

IPv6 uses Network Discovery Protocol (NDP) to find the MAC address. The traditional attacks for exploiting ARP are also relevant in NDP as it is also stateless and lacks authentication of its messages by default. Neighbor Solicitation (NS) spoofing, Neighbor Advertisement (NA) spoofing, Router Solicitation (RS) spoofing, Router Advertisement (RA) spoofing, Neighbor Unreachability Detection(NUD), Duplicate Address Detection(DAD), Router Redirect attack etc. are examples of NDP related attacks. Although there are various attack detection and prevention mechanisms available for ARP attacks (in IPv4 discussed earlier), they are not yet implemented in NDP (IPv6) as the protocol is relatively new and coming in use. A few mechanism has been proposed for detection / prevention of these attacks, but they are either computationally expensive or requires management of cryptographic keys or involve change in NDP itself.

The NS and NA spoofing in NDP is similar to Request and Response spoofing in ARP. Moreover there are additional attack DAD which is also undetectable by both signature and anomaly IDSs for similar reason. Like ARP, IP-MAC pairing information from NS and NA packets are accepted without any verification, attackers can easily spoof NS and NA packet with falsified IP-MAC pairings. NS/NA spoofing involves a malicious node sending NS/NA messages to a target node having falsified IP-MAC pairings. This results in redirecting all the data link layer frames to the spoofed MAC address.

In IPv6, when a new node wants to come up in the network, before using the IP address, it verifies if there is another node which is using the same IP address or it is contending for the same address; this is called DAD attempt. If no such node is found, the new node can use the IP address. Otherwise, another alternative IP address is tried or static IP address is configured [37]. The concept that reply of a node stating "it is using the same IP address or it is contending for the same address" is not verified, can be used by an attacker to launch DoS attack. If the attacker responds to every DAD attempt made by an entering host, then the host will never be able to obtain an address. The attacker can claim the address in two ways:

(1) Attacker can either reply with an NS packet, simulating that it is also performing DAD for same address.

(2) Attacker can reply with an NA packet, simulating that it is already using the address.

Various preventive mechanisms for NDP based attacks have been proposed which

are mentioned as follows. IPsec AH [38] can be used with NDP messages to enhance security and verify through AH that NA contain proper and accurate information. Security Associations (SA)s can be created only by using the Internet Key Exchange (IKE)[39]. But IKE requires a functional IP stack in order to work and this result in a bootstrapping problem. So SA can only be configured manually, which is a tedious or impractical task considering the volume. Even if SAs are established, it is not possible to verify the ownership of dynamically generated IP addresses. Secure Neighbor Discovery (SEND) [40] defines this mechanism which uses PKI to sign NDP messages. Besides, key management in LAN environment is a difficult and cumbersome work for a small or medium scale organization. In addition, cryptographically generated addresses (CGA)[41] are used to avoid spoofing on the local network. However, this protocol is not yet widely implemented and the overhead associated with it can cause DoS conditions itself. Neighbor Discovery Protocol Monitor (NDPmon)[42], is a tool working with ICMPv6 packets, observes the local network to see if nodes using neighbor discovery messages behave properly. When it detects a suspicious Neighbor Discovery message, it notifies the administrator by writing in the syslog. The problem with this approach is, if the first sent packet itself is having a spoofed MAC address then the whole system fails. Further, any genuine change in IP-MAC pair will be discarded.

NDP related attacks are similar to ARP based attacks and also lacks clear signature or anomaly behavior. So active DES framework [21] can be applied to detect these attack. It may be noted that in all the DES frameworks discussed earlier modeling of the normal and attack conditions is done manually, which is assumed to be correct. The effectiveness of the DES based IDSs are dependent on the DES modeling. Specification of NDP and the related attacks are more complex compared to their counterpart in ARP. So if active DES framework is applied to design the IDS for NDP attacks, there are chances of compromise in effectiveness due to the manual modeling step involved.

To address the issue of manual modeling in state based DES framework [20, 21], Jiang et al. in [43] proposed Linear Temporal Logic (LTL) based DES paradigm. As in case of state based models, in the LTL-based DES framework also, first the system under normal and attack conditions are developed manually. Following that, the normal system specification is expressed as LTL formulas. The LTL formulas are then converted into a Finite State Automata (FSA) called Buchi Automata using automated tools [44]. Although, modeling

and specification conversion to LTL formulas are done manually, automated methods like model checking are used to verify the LTL specifications vis-a-vis the system model [66]. Correctness of the LTL formulas and the system model can be determined before progressing to detector design. The detector in LTL based DES paradigm is obtained by proposition synchronization of Buchi automata and the system model. The detector observes system traces and detects a fault if the trace violates the LTL formula.

So for designing IDS for NDP attacks, the LTL based DES framework is used. The first step is modeling of the network under normal and attack(s) condition following by expressing the normal behavior of NDP as a LTL formula. In the next step, the non-faulty behavior of NDP is represented using Buchi Automata, which is obtained automatically from the LTL specification. Correctness of the LTL specification and the system model is next verified. Finally, DES detector based IDS is obtained and it is verified for all traces which violate the normal behavior of NDP.

The contribution of the scheme is enumerated below

1. An IDS to detect NDP related attacks using LTL based DES framework is designed. In most of the design steps of LTL based DES framework FSAs are involved. To handle the state explosion problem, the framework is augmented with model variables.

2. The IDS designed using LTL based DES framework does not require change in protocol, generates minimal extra traffic overhead, resource overhead is not significant and does not require patching in individual host.

3. The proposed model is verifiable thereby making it suitable to model complex system. Verifiable model also ensures that all possible cases of attacks are detected accordingly chances of false positive are reduced.

1.3 Organization of the thesis

The aim of this thesis is to provide detection mechanism for prevalent attacks which does not have signature pattern or anomaly profile. Different class of attacks are considered from different communicating layers like ARP spoofing attacks from Link layer, ICMP based attacks from Network layer, TCP low rate DoS attacks from Transport Layer. This has enabled the study of feasibility of the basic idea of the thesis i.e., “use of DES paradigm

for developing IDS" across layers. Considering the rapid migration from IPv4 to IPv6, IPv6 NDP related attacks are also considered in this work. It may be noted that except Link layer protocols like NDP, attacks (and thus detection mechanisms) pertaining to other layers can be mapped from IPv4 to IPv6.

Rest of the thesis is organized as follows.

Chapter 2: In this chapter an Active DES based IDS for ARP based attacks has been proposed.

Chapter 3: In this chapter IDS for detecting ICMP related attacks is proposed. To deal with the uncertainty in attack detection (non receipt of probe replies due to congestion), I-diagnosis framework[20] of DES is adopted to diagnose ICMP related attacks.

Chapter 4: This chapter presents a novel technique for detecting Low rate TCP attack using stochastic DES.

Chapter 5: This chapter proposes LTL based DES for designing an IDS to detect NDP attacks on IPv6 networks.

Chapter 6: This chapter summarizes the contributions in this thesis and provides direction to future work in this area.

Chapter 2

Active DES based detection of ARP related attacks

2.1 Introduction

In the seven-layer OSI model of computer networking, the data link layer is layer 2. In TCP/IP reference model, it is part of the link layer. This layer provides the functional and procedural means to transfer data between network entities or hosts. Among the layer 2 protocols like Ethernet, Frame Relay, ATM, PPP, Ethernet is the most widely used protocol. In the initial design of various protocols, emphasis is given to performance but not the security issues that might involve into them. As a result, these protocols are susceptible to many attacks which exploit their design weakness, features, loopholes etc.

In layer 2 Ethernet network, there are various attacks which can posse serious security threats. Some of the widely exploited layer 2 attacks having high impact are ARP based attacks, VLAN attacks, Spanning tree attacks, DHCP based attacks etc. Among the various attacks in layer 2, ARP based attacks are one of the widely occurring and single largest attack on Data Link Layer. Many other attacks like MiTM, DoS etc. are launched by sending falsified IP-MAC pairs to a host being targeted. This motivated us to study these group of attacks and propose a comprehensive detection mechanism for the class of attacks. In case of ARP attack, victim host assumes the MAC address in the forged ARP packet as the genuine MAC address associated with the IP. Now, when the victim host wants to communicate with the system having the given IP, it is forced to send all packets to the

false MAC address (i.e., to a different host the attacker wants the victim to send).

In layer 2, hosts are identified by a 48 bit address known as MAC address where as hosts in the Internet use 32 bit IP address (IPv4) to identify hosts. When two hosts want to communicate with each other, they need to know the MAC address of each other. Address Resolution Protocol (ARP) is used by hosts to map network addresses known as IP address to physical addresses called MAC address. If the communicating host does not know MAC address of the destination, it sends an ARP request to the broadcast domain asking for the MAC address corresponding to the destination host's IP address. The destination host identifies that the ARP request is meant for its IP address and hence, sends back its MAC address in a unicast ARP reply packet. Attacks based on falsified IP-MAC pairs are feasible because host updates its ARP cache without verifying the genuineness of the source. Also, the hosts cache all the ARP replies sent to them even if they had not sent an explicit ARP request.

In ARP attacks, attacker crafts and sends fake ARP packets to victim host and thereby, poisoning victim's ARP cache. This is called *ARP spoofing* [45]. ARP spoofing leads to many other attacks. In the *ARP based Man-in-the-Middle(MiTM) Attack*, the attacker spoofs the ARP packets and inserts itself between the communication path of two target computers. This allows the attacker to sniff all the packets transferred between the target systems and paves way for the attacker to modify the messages if it desires. The attacker can also fabricate an ARP packet with non-existent MAC, and successfully cause Denial of Service(DoS) to the victim host. On a switched network, the attacker can send numerous spoofed ARP replies to a switch at an extremely rapid rate. This leads to *ARP Flooding* or *MAC Flooding* where the switch's port/MAC table (CAM table) starts to overflow.

Various mechanisms have been proposed to detect and mitigate these ARP attacks at both the host-level and network-level[8, 9, 10, 12, 13, 46, 47, 48]. However, many of these techniques are not practically possible to implement. Some of them stand against the basic principle of network like scalability, dynamism etc. A few of them even calls for a change in protocol stack. FDD of DES has been adapted for IDS in [16, 17, 18], where attacks are mapped to failures and the diagnoser is implemented as the IDS engine. The attack detection capability of these IDSs are based on passive monitoring of sequence of events with the assumption that intrusions lead to change in the sequence (which needs to be detected). However for ARP based attacks, passive monitoring schemes have

several limitations because in such attacks there is no change in sequence of events. IDSs with active probing are now being proposed for such attacks which involve sending of probe packets that cause difference in sequence of events under attack condition and can be detected using passive monitoring. To the best of our knowledge no state-transition framework with active properties (i.e., transitions being controllable/uncontrollable) has been applied in development of such IDSs. In this chapter we propose an IDS to detect ARP spoofing attacks using active state-transition framework called active DES.

Rest of the chapter is organized as follows. In Section 2.2 the ARP attacks and the existing schemes in literature to defend or detect against these attacks are discussed. The motivation for building the DES based detection approach is also noted here. Section 2.3 describes the proposed DES based approach to combat this attack. The adapted active DES framework is also discussed. Section 2.5 describes the experimental setup for implementing the IDS and discusses the results. Finally we conclude in Section 2.6.

2.2 ARP based attacks and existing countermeasures

ARP is a stateless protocol. There is no mechanism to authenticate or validate the response and this is the major loop hole in the protocol. Hence, there is no means to prevent the hosts from crafting custom ARP packets by which it can forge the IP and MAC addresses. The various ARP based attacks are discussed next.

2.2.1 Attacks Based on ARP

2.2.1.1 ARP Spoofing

This is a technique whereby an attacker sends fake(spoofed) ARP messages onto a network. Generally, the aim is to associate the attacker's MAC address with the IP address of another host (such as the default gateway), causing any traffic meant for that IP address to be sent to the attacker instead. When a malicious user sends falsified responses to a target system, it blindly updates its cache with the IP-MAC pairing in the response packet. Using this technique, the attacker can send forged packets to the target systems and poison their caches. This is the process of ARP spoofing. ARP spoofing is the basic attack which acts as a gateway for many network-specific attacks.

2.2.1.2 ARP MiTM Attack

When an MiTM is performed, the attacker inserts itself between the communication path of two target computers. It achieves this by sending forged ARP packets to both the targets and hence poisoning their caches. Sniffing can then be performed. The malicious computer will then forward the frames between the two target computers so that the communication is not interrupted.

Once MiTM attack is launched between the hosts, sniffing and session hijacking may easily be performed. Session hijacking enables an attacker to take control of a connection between two computers. For instance, an attacker could take control of a telnet session after a target computer has logged into a remote computer as administrator.

2.2.1.3 ARP DoS

DoS to a host: When the attacker fabricates an ARP packet such that the IP address of a legitimate host is mapped to a non-existent MAC, all the packets addressed to this particular host are lost (as packets are sent to non-existent MAC address).
DoS to the gateway: When the IP address of the gateway is mapped to a non-existent MAC, systems fails to connect to the gateway. This might result in a serious issue. The malicious user can utilize this attack to block all the peer systems to connect to the Internet, and instead utilize the whole bandwidth for its own purpose.

2.2.1.4 MAC Flooding

On a switched network, MAC flooding is a serious issue. By sending spoofed ARP replies to a switch at an extremely rapid rate, the switch's port/MAC table (CAM table) will overflow. At this point, switches may revert to the broadcast mode. As a result, sniffing can be performed easily.

There are numerous tools available which make it very simple to perform these ARP-based attacks. The tools are Ettercap, Cain and Abel, Dsniff, Parasite, ArpPoison, and ARP-SK etc. Since these tools are quite simple and easy to operate, the ARP attacks are becoming more common these days even among normal computer users. These attacks can not be detected by the operating systems and most firewalls since they do not implement the lower layer security.

2.2.2 Existing Detection Mechanisms of Attacks based on ARP

Various countermeasures and detection mechanisms for detecting ARP based attacks with their drawbacks are as follows.

2.2.2.1 Static ARP Entries[8]

The most foolproof way to prevent ARP attacks is to manually assign static IPs to all systems and maintain the static IP-MAC pairings at all the systems. However, this scheme is not suitable for a dynamic environment.

2.2.2.2 Security Features[9]

One possible action to combat ARP attacks is enabling port security on the switch. This is a feature available in high-end switches which tie a physical port to a MAC address. These port-address associations are stored in Content Addressable Memory (CAM) tables. A change in the transmitter's MAC address can result in port shutdown or ignoring the change. The problem with this approach is, if the first sent packet itself is having a spoofed MAC address then the whole system fails. Further, any genuine change in IP-MAC pair will be discarded (e.g., when notified by Gratuitous request and reply).

2.2.2.3 Software based solutions

The basic notion of port security involving observation of changes in IP-MAC pairs in switches has also been utilized in software solutions namely, ARPWATCH [10], COLASOFT-CAPSA [12]. These software solutions are cheaper than switches with port security but have slower response time compared to switches. Obviously, these tools suffer from the drawbacks as that of port security in switches.

2.2.2.4 Signature and anomaly based IDS

Signature based IDSs like Snort [4] can be used to detect ARP attacks and inform the administrator with an alarm. However, the main problem with these IDSs in case of ARP attacks is the number of false positives. Furthermore, ability of IDSs to detect all forms of ARP related attacks is limited [13]. Recently, Hsiao et al. [46], have proposed an anomaly

IDS to detect ARP attacks based on SNMP statistics. A set of features are extracted from SNMP data and data mining algorithms such as decision tree, support vector machines and bays classifier have been applied to classify attack data from normal data. Reported results show that false positive rates are as high as 40%.

2.2.2.5 Modifying ARP using cryptographic techniques

Several cryptography based techniques have been proposed to prevent ARP attacks namely S-ARP[47], TARP [48]. Addition of cryptographic features in ARP lead to performance penalty [13] and change the basic ARP.

2.2.2.6 Active techniques for detecting ARP attacks

An IDS using active detection for ARP attacks, sends probe packets in addition to observations in changes of IP-MAC pairs.

In [14], a database of known IP-MAC pairs is maintained and on detection of a change the new pair is actively verified by sending a probe with TCP SYN packet to the IP under question. The genuine system will respond with a SYN/ACK or RST depending on whether the corresponding port is open or closed. While this scheme can validate the genuineness of IP-MAC pairs, it violates the network layering architecture.

An active scheme for detecting MiTM attack is proposed in [15]. The scheme assumes that any attacker involved in MiTM must have IP forwarding enabled. First, all systems with IP forwarding are detected (actively). Then the IDS attacks all such systems one at a time and poison their caches. The poisoning is done in a way such that all traffic being forwarded by the attacker reaches the IDS (instead of the system, the attacker with IP forwarding wants to send). In this way the IDS can differentiate real MiTM attackers from all systems with IP forwarding. There are several drawbacks in this approach, namely huge traffic in case of a large network with all machines having IP forwarding, assumption of successful cache poisoning of the machine involved in MiTM attack, cache poisoning (of the machine involved in MiTM attack by IDS) exactly when the attack is going on etc.

From the review, it may be stated that an ARP attack preventation/detection scheme needs to have the following features.

- Should not modify the standard ARP.

- Should generate minimal extra traffic in the network.
- Should not require patching, installation of extra softwares in all systems.
- Should detect a large set of ARP based attacks.
- Hardware cost of the scheme should not be high.

Discrete Event System (DES) based framework has been widely used for Failure Detection and Diagnosis (FDD) because of modeling simplicity and the associated algorithms [19]. Further, most of the systems can be modeled as DES using some level of abstraction. A DES is characterized by a discrete state space and event driven dynamics. The basic idea is to develop a DES model for the system under normal condition and also under each of the failure conditions. Following that, a state estimator called diagnoser (or detector, if only detection of failure is required) is designed which observes sequence of events generated by the system to decide whether the states through which the system traverses correspond to the normal or faulty DES model. FDD of DES has been adapted for IDS in [16, 18, 17], where attacks are mapped to failures and the diagnoser is implemented as the IDS engine.

Looking into the advantages of active detection mechanisms for ARP attacks compared to their passive counter parts, Neminath et al. in [17, 18] have proposed a DES based IDS which uses active probes. However, in spite of the IDS being active the DES framework used [20] is a passive one. So, in their approach, the probing module is not a part of the formal DES framework. Only, responses to probes are formally modeled as DES and IDS is constructed from the DES detector. The main advantage of using a formal framework like DES is integrity, verifiability etc. In this chapter we propose an IDS for ARP attacks based on FDD theory of *active* DES framework [21], which enables modeling of all ARP related network events (including active probes) in the formal framework. In active framework, the probes are modeled as controllable events as they are sent by IDS to verify IP-MAC pairs. The responses to the probes are modeled as uncontrollable events, which are monitored by the detector to determine whether attack has taken place. To the best of our knowledge no formal framework with active properties (i.e., transitions being controllable/uncontrollable) has been applied in development of IDS.

2.3 Proposed Active DES based IDS for ARP related attacks

In this section the proposed DES based intrusion detection scheme for ARP related attacks is presented.

The following assumptions are made.

- Non-compromised hosts will send one response to an ARP request within a specific interval T_{req}.
- IDS is running on a dedicated and trusted machine with a fixed IP.
- Port mirroring is enabled at the switch so that IDS has a copy of all outgoing and incoming packets from all ports.

In the next subsection the probing technique is discussed. Following that an example is given to motivate the requirement of probing to generate different sequence of ARP packets under normal and attack scenario.

2.3.1 Active Probing Technique

Upon receiving ARP requests or ARP replies, ARP probes are sent to the source IP of the request or reply packet. Before sending probes it is checked that the source IP-MAC pair is not yet verified by the DES detector to be genuine or spoofed. Details of such verified IP-MAC pairs are recorded in tables, discussed next. Henceforth in the discussion, short notations are used as shown in Table 2.1

Table 2.1: Notations used

Short Notation	Description
IPS	Source IP Address
IPD	Destination IP Address
MACS	Source MAC Address
MACD	Destination MAC Address
RQP	ARP request packet
RSP	ARP response packet
RQP_{IPS}	Source IP of *RQP*;
PRQP	Probe request packet
PRSP	Probe response packet

Similar to RQP_{IPS}, subscripting will be used to denote destination IP, source MAC and destination MAC for *RQP* and *RSP*.

1. Every time an ARP request is verified to be genuine an entry is made in Authenticated table, denoted as *AUTHT*. It has five fields, source IP $AUTHT_{IPS}$, source MAC $AUTHT_{MACS}$, destination IP $AUTHT_{IPD}$, destination MAC $AUTHT_{MACD}$ and time when the request packet is received by IDS $AUTHT_{Ti}$.

2. For spoofed requests the details are entered in another table called Spoofed table (denoted as *SPOOFT*) which has same fields as Authenticated table.

3. Every time an ARP response is verified to be genuine (or spoofed) an entry is made in the Authenticated (Spoofed) table.
   In case of ARP requests only four out of five fields in the tables can be filled; *MACD* left as " − −" as no destination MAC address is associated with requests. However, in case of responses all the five fields are filled with appropriate values.

$\langle TableName \rangle_{MAX}$ represents the maximum number of elements in a table at a given time. These tables not only help in minimizing ARP probes but is also used for determining if spoofing has led to other attacks like MiTM, DoS etc.

The probing technique has two main modules namely, ARP REQUEST-HANDLER() and ARP RESPONSE-HANDLER(). These are elaborated in Algorithm 2.1 and Algorithm 2.2, respectively. Algorithm 2.1 process ARP request packets in the network. For any ARP request packet *RQP*, it first verifies if it is malformed (i.e., any changes in the immutable fields of the ARP packet header or different MAC addresses in the MAC and ARP header field) or unicast. These cases are abnormal and decision can be made on observation of any such (single) packet, unlike sequence of ARP packets in case of spoofing. Detection of such abnormal cases are trivial and we do not consider them in DES modeling and attack detection. Algorithm 2.1 simply exits (by setting a Status flag) on such ARP request packets. If the packet is not unicast or malformed, but a request packet from (IDS) i.e., RQP_{IPS} is IP of IDS and RQP_{MACS} is MAC of IDS, Algorithm 2.1 skips processing of this packet; we need not send ARP probes to ARP probe request from the IDS as we assume that IP-MAC pairing of the IDS is known and validated. If none of the above cases are matched, then RQP_{IPS} is searched in the Authenticated table. If a match is found as $AUTHT_{IPS}[i]$ and

the corresponding source MAC address $AUTHT_{MACS}[i]$ in the table is same as RQP_{MACS}, the packet has genuine IP-MAC pair which is already determined by the detector. *This genuineness is obtained without explicit use of the DES detector. Also, ARP probes are not sent in this case thereby saving some ARP traffic.* RQP_{IPS}, RQP_{MACS}, RQP_{IPD}, $--$ *and Time of receipt are stored in the Authenticated table.* In case of a match in the source IP and mismatch in the source MAC address (i.e., $RQP_{IPS} = AUTHT_{IPS}[i]$ and $RQP_{MACS} \neq AUTHT_{MACS}[i]$) the packet is detected to be spoofed *without explicit use of the DES detector and no ARP probes are sent.* The details of the *RQP* are stored in the Spoofed table. On the other hand if RQP_{IPS} is not found in *AUTHT* (in field $AUTHT_{IPS}$), then both RQP_{IPS} and RQP_{MACS} are searched in *SPOOFT* (i.e., $RQP_{IPS} = SPOOFT_{IPS}[i]$ and $RQP_{MACS} = SPOOFT_{MACS}[i]$). If a match is found then it implies that the IP-MAC pair is already detected to be spoofed. No ARP probes are sent and details of the request packet are added in the *SPOOFT*. If both the above cases are not satisfied then an ARP probe request is sent (broadcast) by Algorithm 2.1 requesting the *MAC* for the source IP RQP_{IPS}. In another case i.e., gratuitous ARP requests, ARP probe is sent. Gratuitous ARP requests are broadcast (e.g., entry of a new host or change in NIC card) to inform other hosts about its IP-MAC pair. Gratuitous ARP request can be determined if $RQP_{IPS} = RQP_{IPD}$. For such Gratuitous request, ARP probe is sent to the RQP_{IPS} without verifying in the tables because such requests are made when a host with "new IP-MAC pair" comes up which needs to be verified by the detector afresh.

As already discussed, any DES model has states, and transitions are made among states on occurrence of some (discrete) events. These events are modeled based on certain changes in the system. For example, in a chemical reaction chamber, temperature moving above threshold can be modeled as an event which leads to change in state to make heater off [19]. Sensors are kept in the system to measure such changes, e.g., a thermocouple/temperature sensor in case of the chemical chamber. In our case, we consider the following changes as events: Receipt of *RQP*, sending of ARP probe request (*PRQP*), receipt of ARP probe response (*PRSP*), receipt of response (other than for ARP probe) *RSP*. These requets/responses with their respective fields (e.g., source IP of *RSP*) are considered as measured events of the DES model. In addition, when Algorithm 2.1 decides some IP-MAC pair to be geniune/spoofed using Authenticated table/Spoofed table, it intimates the same; this is captured by event "detected (*DTD*)".

Algorithm 2.2 is ARP response handler. For any ARP reply packet *RSP*, the algorithm

Algorithm 2.1 ARP REQUEST HANDLER

Input : *RQP* - ARP request packet

Output: Intimate receipt of *RQP*, sending of ARP probe request and detected *DTD*, Updated *AUTHT* and *SPOOFT*

if (*RQP* is *malformed*) OR (*RQP* is *Unicast*) **then**
  "Status is abnormal" and EXIT
end if
if (RQP_{IPS} = *IP*(*IDS*)) AND (RQP_{MACS} = *MAC*(*IDS*)) **then**
  EXIT
end if
if (RQP_{IPS} =$AUTHT_{IPS}$[*i*] AND RQP_{MACS} = $AUTHT_{MACS}$[*i*], for any *i*, $1 \leq i \leq AUTHT_{MAX}$) **then**
  "Status is Genuine";
  add {RQP_{IPS}, RQP_{MACS}, RQP_{IPD}, −−, time of receipt} in *AUTHT*;
  Intimate *RQP* and *DTD*; EXIT
end if
if (RQP_{IPS} =$AUTHT_{IPS}$[*i*], for any *i*, $1 \leq i \leq AUTHT_{MAX}$) AND ($RQP_{MACS}$! = $AUTHT_{MACS}$[*i*]) **then**
  "Status is Spoofed";
  add {RQP_{IPS}, RQP_{MACS}, RQP_{IPD}, −−, time of receipt} in *SPOOFT*;
  Intimate *RQP* and *DTD*; EXIT
end if
if (RQP_{IPS} =$SPOOFT_{IPS}$[*i*] and RQP_{MACS} =$SPOOFT_{MACS}$[*i*], for any *i*, $1 \leq i \leq SPOOFT_{MAX}$) **then**
  "Status is Spoofed";
  add {RQP_{IPS}, RQP_{MACS}, RQP_{IPD}, −−, time of receipt} in *SPOOFT*;
  Intimate *RQP* and *DTD*; EXIT
else(/* Gratuitous i.e., RQP_{IPS} == RQP_{IPD} or none of the above conditions */)
  Intimate receipt of *RQP*;
  Send ARP Probe Request *PRQP* to RQP_{IPS} from IDS;
  Intimate sending of the ARP probe request *PRQP*;
end if

first finds the malformed packets and sets the status accordingly. Next, it verifies whether the reply packet is for any ARP probe (sent by the IDS). The response for an ARP probe can be determined if RSP_{IPD} is IP of IDS and RSP_{MACD} is MAC of IDS. For these packets no probe is sent and the algorithm intimates the receipt of these response packets (*PRSP*). If the response is neither malformed nor reply to an ARP probe, the IDS sends an ARP probe request (broadcast) to the source IP RSP_{IPS} requesting its *MAC*. Before sending the probe, Algorithm 2.2 does a check in the Authenticated table and Spoofed table, similar to the one discussed in Algorithm 2.1.

Algorithm 2.2 ARP RESPONSE HANDLER()

Input : *RSP* - ARP response packet
Output: Intimate receipt of *RSP*, ARP probe response *PRSP*, sending of ARP probe request *PRQP* and detected *DTD*, Updated *AUTHT* and *SPOOFT*

if (*RSP* is *malformed*) **then**
  "Status is abnormal" and EXIT
end if
if (RSP_{IPD} = *IP*(*IDS*)) AND (RSP_{MACD} = *MAC*(*IDS*)) **then**
  Intimate receipt of *PRSP*; EXIT
end if
if (RSP_{IPS} =$AUTH_{IPS}[i]$ AND RSP_{MACS} = $AUTH_{MACS}[i]$, for any i, $1 \leq i \leq AUTHT_{MAX}$) **then**
  "Status is Genuine";
  add {RSP_{IPS}, RSP_{MACS}, RSP_{IPD}, RSP_{MACD}, time of receipt} in *AUTHT*;
  Intimate *RSP* and *DTD*; EXIT
end if
if (RQP_{IPS} =$AUTH_{IPS}[i]$, for any i, $1 \leq i \leq AUTH_{MAX}$) AND (RQP_{MACS}! = $AUTH_{MACS}[i]$) **then**
  "Status is Spoofed";
  add {RSP_{IPS}, RSP_{MACS}, RSP_{IPD}, RSP_{MACD} and time of receipt} in *SPOOFT*;
  Intimate *RSP* and *DTD*; EXIT
end if
if (RQP_{IPS} =$SPOOFT_{IPS}[i]$ and RQP_{MACS} =$SPOOFT_{MACS}[i]$, for any i, $1 \leq i \leq SPOOFT_{MAX}$) **then**
  "Status is Spoofed";
  add {RSP_{IPS}, RSP_{MACS}, RSP_{IPD}, RSP_{MACD}, time of receipt} in *SPOOFT*;
  Intimate *RSP* and *DTD*; EXIT
else(/* Gratuitous i.e., RSP_{IPS} == RSP_{IPD} or none of the above conditions */)
  Intimate receipt of *RSP*;
  Send ARP Probe Request *PRQP* to RSP_{IPS} from IDS;
  Intimate sending of the ARP probe request *PRQP*;
end if

2.3.1.1 An Example

An example is used to illustrate handling of spoofed response packets by Algorithm 2.2. Further, this example also highlights the difference in ARP packet sequence (after active probing) in case of spoofing versus normal senecio. Here, the network has four hosts A, B, D and E; E is the IDS and D is the attacker. The machines A,B,D and E are assigned IP addresses 10.0.1.1, 10.0.1.2, 10.0.1.4 and 10.0.1.5 respectively. The MAC addresses of the hosts A, B, D and E are 08:4D:00:7D:C1, 08:4D:00:7D:C2, 08:4D:00:7D:C4 and 08:4D:00:7D:C5, respectively. Port mirroring is enabled at the switch so that E has a copy of all outgoing and incoming packets from all ports. Also, E has a network interface to solely send ARP probes and receive ARP probe responses.

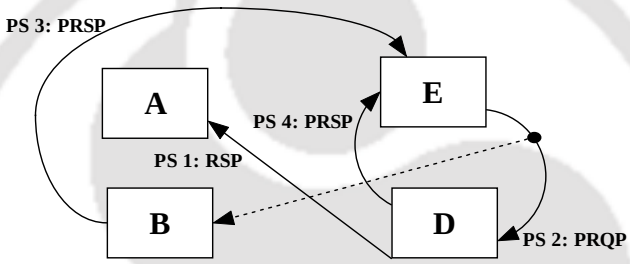

Figure 2.1: Example of Spoofed Response

Figure 2.1 shows the sequence of packets (indicated with packet sequence number) injected when attacker D is sending a spoofed reply as "IP(B)-MAC(D) " to host A and its verification (using active probe). As packet sequence 1, D sends an ARP response *RSP* to A telling that the IP of B is associated with MAC of D. This ARP response conversation is viewed in tcpdump (one of the utilities for watching packets visible to an interface) as:
08:4D:00:7D:C4 08:4D:00:7D:C1 60: ARP reply 10.0.1.2 is-at 08:4D:00:7D:C4.
As ARP is a stateless protocol, after receiving the response from D, A updates its cache with IP-MAC pair as IP(B)-MAC(D); the ARP cache of A with the update is shown in Table 2.2.

Table 2.2: ARP Cache of Machine A Under Spoofed Reply

IP	MAC	Interface
10.0.1.2	08:4D:00:7D:C4	eth0

It may be noted that under normal condition "B's reply to A with IP-MAC pair :IP of B-MAC of B" and attack condition "D's reply to A with IP-MAC pair :IP of B-MAC of D",

there is no change in sequence of ARP packets. Under the attack, all traffic A wants to send to B will be sent to D.

Now the use of active probing is discussed that creates a difference in sequence of ARP packets under normal and attack situation. Initially ARP cache table of all the machines are empty. Also, it is assumed that the Authenticated table and Spoofed table are empty (at this instant).
On receiving the *RSP* from D to A (packet sequence 1), ARP RESPONSE HANDLER() at E, intimates the receipt of *RSP* (event 1), sends ARP probe request *PRQP* to IP of B to query the corresponding MAC (packet sequence 2) and intimates sending of the ARP probe request *PRQP* (event 2). This ARP probe request conversation is viewed in tcpdump as:
08:4D:00:7D:C5 ff:ff:ff:ff:ff:ff 42: ARP who-has 10.0.1.2 tell 10.0.1.5.
Probe request is sent, as the source IP-MAC pair of this *RSP* is not yet verified (Authenticated table and Spoofed table being empty). As ARP probe is a broadcast, both B and attacker D will get the probe. B will definitely respond to the probe as *PRSP* with IP(B)-MAC(B) to E (packet sequence 3) as it is assumed that the non-comprised host (B) will always respond to the probe. This ARP probe response conversation is viewed in tcpdump as:
08:4D:00:7D:C2 08:4D:00:7D:C5 60: ARP reply 10.0.1.2 is-at 08:4D:00:7D:C2.

Now the IDS can know that the response made in packet sequence 1 is false (as it already has IP(B)-MAC(D)) and the IDS can generate an alarm (and also track the attacker MAC (D)). To avoid self identification, attacker D has to reply to all queries for MAC of IP(B) with IP(B)-MAC(D); this ARP probe response from D is viewed in tcpdump as:
08:4D:00:7D:C4 08:4D:00:7D:C5 60: ARP reply 10.0.1.2 is-at 08:4D:00:7D:C4.
It may be noted that the order of reply by the attacker and the genuine host is not fixed. So, if both attacker and genuine host reply to a request with their MACs, the IDS can detect a spoofing but cannot point the attacker's MAC. So, it is reasonable to believe that an attacker would reply to all queries against the IP it is spoofing. To summarize, at least one reply to a probe (send to verify IP(B), say) expected to arrive and have genuine IP-MAC of B. In case of spoofing, more replies expected to arrive which may have different MACs. Once the spoofing is detected an entry is made in Spoofed table as shown in Table 2.3.

In this example, we assume that the genuine host B replies (packet sequence 3) before

Table 2.3: Spoofed Table Entry When Spoofing is Detected

SRC IP	SRC MAC	Dest IP	Dest MAC	Time
10.0.1.2	08:4D:00:7D:C4	10.0.1.1	08:4D:00:7D:C1	00:00:30

attacker D (packet sequence 4) to the ARP probe. The replies in packet sequence 3 and packet sequence 4 are processed by ARP RESPONSE HANDLER() as "Intimate receipt of *PRSP*" thereby generating event 3 and event 4, respectively. It may be noted that in the two replies, different MACs are associated with the IP being probed i.e., once MAC(B) is associated with IP(B) and then MAC(D) is associated with IP(B). Table 2.4 lists the sequence of ARP packets for the example (in Figure 2.1).

Table 2.4: Tabulation of the packet sequence and events in the example

PS: Events	SRC IP	SRC MAC	Dest IP	Dest MAC
PS 1:*RSP*	10.0.1.2	08:4D:00:7D:C4	10.0.1.1	08:4D:00:7D:C1
PS 2:*PRQP*	10.0.1.5	08:4D:00:7D:C5	10.0.1.2	–
PS 3:*PRSP*	10.0.1.2	08:4D:00:7D:C2	10.0.1.5	08:4D:00:7D:C5
PS 4:*PRSP*	10.0.1.2	08:4D:00:7D:C4	10.0.1.5	08:4D:00:7D:C5

Now let us see the sequence of ARP traffic if the response in packet sequence 1 is genuine, i.e., IP(B)-MAC(B) is sent to A by B. For the ARP probe sent to verify IP(B)-MAC(B) (packet sequence 2) only B will respond to E with IP(B)-MAC(B) (packet sequence 3). Till T_{req} time the IDS will receive only one *PRSP* (unlike two in case of attack). Once the DES detector (i.e., IDS) finds the reply to be genuine (by virtue of only one *PRSP*), details of *RSP* are stored in the Authenticated table. If more than one *PRSP*s are received within T_{req} with different MACs, details of *RSP* are stored in the Spoofed table. Following that, no probes will be sent for any ARP reply/request with source IP as IP(B). However, in case of a Gratuitous request/reply, ARP probe will be sent.

As discussed before, the IDS engine for detecting the ARP spoofing attacks can be constructed using a DES detector. So, initially the ARP events under normal and spoofed conditions are modeled using state-event based DES models and subsequently a DES detector is designed. Before we discuss the formal DES modeling framework we illustrate abstract state-event based models for ARP under normal and spoofed conditions. Figure 2.2 shows the state based ARP model under normal condition. Events listed before, namely receipt of *RQP*/*RSP*, sending of ARP probe request (*PRQP*) and receipt of ARP

probe response (*PRSP*) are shown in bold with the transitions in the figure. Further, some elaboration of the events are also illustrated with the transitions. Figure 2.3 (A) and Figure 2.3 (B) show the model under request spoofing and response spoofing, respectively.

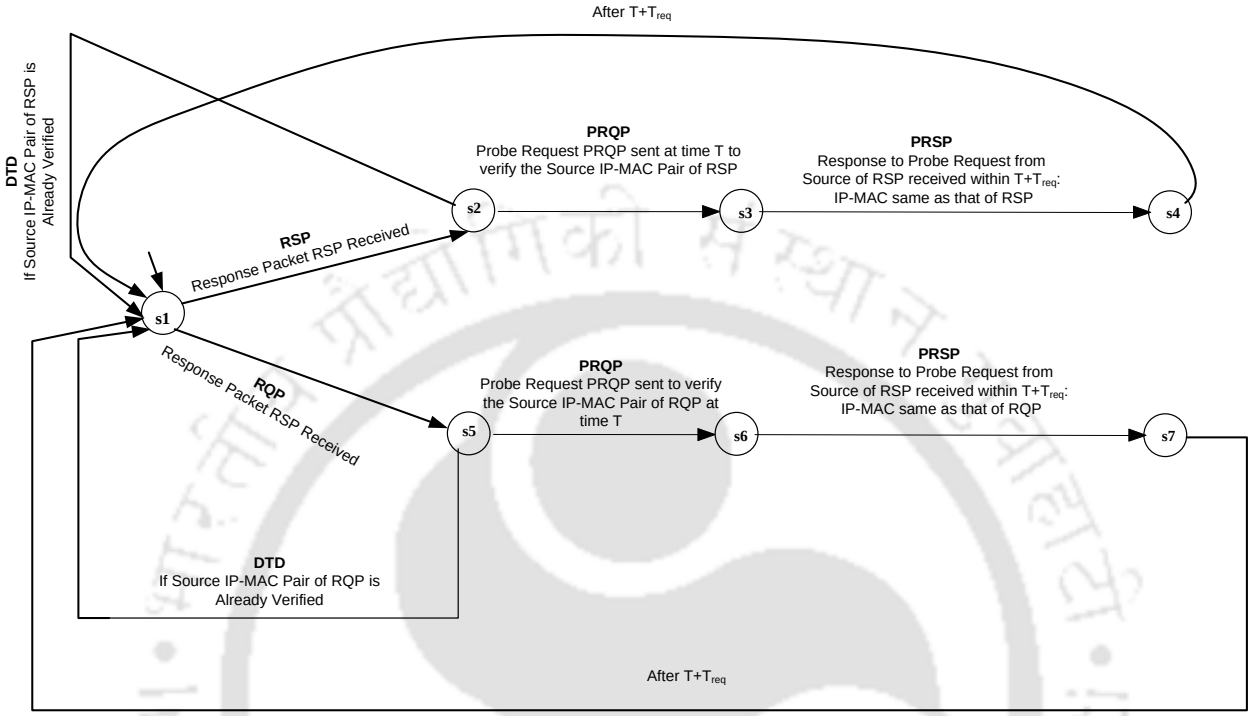

Figure 2.2: State based ARP model Under Normal Condition

2.4 Active DES modeling

In this section we present the use of active DES framework for detection of ARP spoofing attacks. The FDD framework for active DES proposed in [21] has been adapted for ARP attack detection. Extensions over the theory presented in [21] have been made for applying it in development of IDS for ARP attacks. The motivations and the extensions are enumerated next.

1. The size of domains of variables involved in DES modeling for IDS is very large compared to systems usually handled by DES frameworks like pump-valve systems, chemical plants etc. (orders of thousands). On the other hand in the case of IDSs, for example, the size of domain of IP address can be $O(256.256.256.256)$. So in the adapted active DES model apart from state variables model variables are also incorporated.

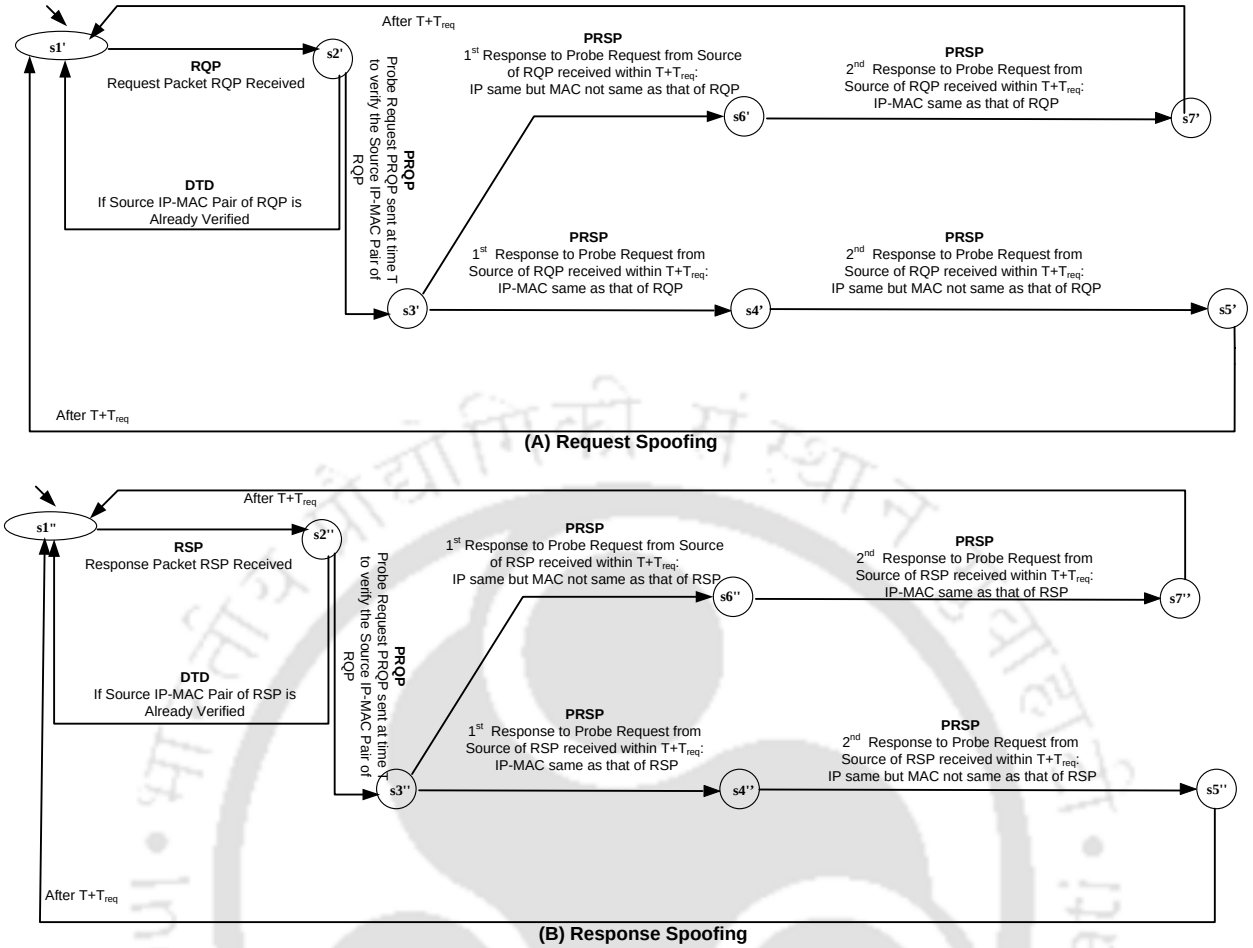

Figure 2.3: State based ARP model Under Request Spoofing and Response Spoofing

In the next subsection we discuss the adapted active DES modeling framework. Following that we present the modeling of ARP spoofing attacks in this active DES framework. Also, design of the DES diagnoser to detect such attacks is disused.

2.4.1 State explosion of DES

In active DES framework, specifications are modeled using Finite State Automata. This means the system and the detector are modeled as a state-transition systems. The detector checks event traces and reports fault if the trace do not meet the specification. Broadly, speaking most of the steps in the active DES based framework are in form of state-transition systems termed as Finite State Automata (FSAs). The major issue with FSAs is that they suffer from the state explosion problem if the system to be modeled is complex. ARP is responsible for mapping IP addresses to MAC addresses, so IP and MAC addresses are the

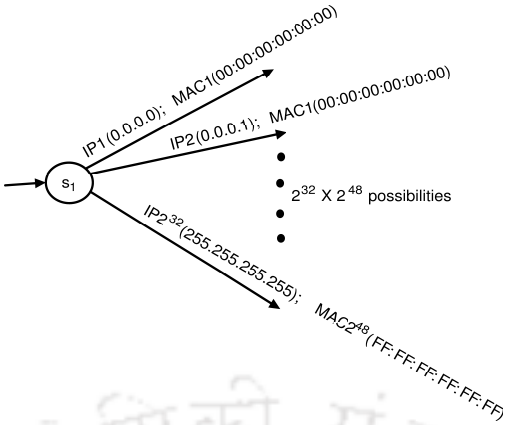

Figure 2.4: State Explosion in System Model

most fundamental parameters that need to be modeled. As the range of IP addresses is $O(2^{32})$-MAC addresses is $O(2^{48})$ and theoretically any IP address can be associated with any MAC address (by the attacker), the FSA model may have extreme large number of states. This would lead to state explosion problem as shown in Figure 2.4. So the active DES based framework cannot be directly applied for modeling ARP and related attacks.

To handle this issue, the active DES framework has been augmented with *model variables*, which preserves the advantages and at the same time address the issue of state explosion. The model variables are associated with each transition, where they can be assigned values and checked for equality for firing the transition. In case of modeling the ARP, we have model variable for IP address and MAC address. Any IP-MAC pairing can be modeled by assigning the values to corresponding model variables, thereby avoiding state explosion.

2.4.2 Adapted Active DES modeling

The active DES model G is a six tuple $\langle \Sigma, X, X_0, V, C, \Im \rangle$, where Σ is the set of events, S is the set of states, $X_0 \subseteq X$ is the set of initial states, V is the set of model variables, C is the set of clock variables and $\Im$ is the set of transitions. It may be noted that there is no final state as the ARP traffic is semi-renewal process (never halts as long as network is up). There are some states from which a transition to an initial state(s) exist, representing renewal of the process; such states are termed as renewal states. So the DES model is live.

Each element v_i of V ($V = \{v_1, v_2, \cdots, v_n\}$) can take values from a domain D_i. The clock variables take values in non-negative reals $\mathbb{R}$. An invariant condition on a clock variable c, denoted as $\Phi(c)$, is a linear inequality or equality of the variable with a non-negative real.

A transition $\tau \in \Im$ is a seven tuple $\langle x, x', \sigma, \phi(V), \Phi(C), Reset(C), Assign(V)\rangle$, where x is the source state, x' is the destination state, σ is an event (on which the transition is fired), $\phi(V)$ is boolean conjunction of equalities of a subset of variables in V, $\Phi(C)$ is the invariant condition on a subset of clock variables, $Reset(C)$ is a subset of clock variables to be reset and $Assign(V)$ is a subset of model variables and assignments with values from their corresponding domains.

A *trace* of a DES model G is a sequence of transitions *generated* by G and denoted as $s \stackrel{\Delta}{=} \langle \tau_1, \tau_2, \cdots \rangle$, where $initial(\tau_1)$ is an initial state in X_0 and the consecution property holds, that is, $initial(\tau_{i+1}) = final(\tau_i)$, for $i \geq 1$. Traces are infinite and a finite prefix of a trace is referred to as a "finite trace". Henceforth, we assume the consecution property for any "sequence of transitions". For any trace $s = \langle \tau_1, \tau_2, \cdots \rangle$, $initial(s) = initial(\tau_1)$ and for a finite prefix $s = \langle \tau_1, \tau_2, \cdots \tau_f \rangle$, $final(\tau_f) = final(s)$. A state x is said to be in a trace s, if $x = initial(\tau_i)$, for some $i \geq 1$. The set of all traces generated by G and their finite prefixes is the language of G, denoted as $L(G)$. The set $L_f(G)$ denotes the subset of $L(G)$ comprising the finite prefixes of the members of $L(G)$. Naturally, $L(G) - L_f(G)$ is a subset of $\Im^w$, where $\Im^w$ is the set of all infinite sequence of $\Im$; $L_f(G)$ is a subset of $\Im^*$, the Kleene closure of $\Im$. The post language of G after a finite prefix s of a trace, denoted as $L(G)/s$, is defined as $L(G)/s = \{t \in \Im^w \cup \Im^* | st \in L(G)\}$. $L_f(G)/s \subset L(G)/s$ comprises finite prefixes of the traces of $L(G)/s$.

The transitions are partitioned into measurable (denoted as $\Im_u$) and un-measurable subsets (denoted as $\Im_{um}$). Also the transactions are classified as controllable (denoted as $\Im_c$) and un-controllable (denoted as $\Im_{uc}$) subsets. The next subsections elaborate measurability and controllability of the DES model.

2.4.2.1 Model with measurability

To capture measurement limitations, the set of transitions are partitioned into two disjoint subsets, $\Im_u$ and $\Im_{um}$, of *measurable* and *unmeasurable* ones, respectively. Example of measurable transitions can be corresponding to events like receipt of ARP request packets, sending probe packets etc. while example of unmeasurable transition can be launching of attack (i.e., failure causing event). It may be noted that launching of an attack cannot be directly observed and needs to be concluded by sequence of other observable transitions. In an unmeasurable transition ($\tau_{um} \in \Im_{um}$) only source state, destination state and triggering

event are defined, i.e., $\tau_{um} : \langle x, x', \sigma, -, -, -, - \rangle$. So enabling of τ_{um} is only dependent on σ and not on any model variables or clock invariant condition; this is depicted by " $-$ " in the transition. Also, τ_{um} does not assign model variables or reset clock variables.

Definition 2.1 (Measurement Equivalent Transitions and States): *Two transitions* $\tau_1 = \langle x_1, x_1', \sigma_1, \phi_1(V), \Phi_1(C), Reset_1(C), Assign_1(V) \rangle$ *and* $\tau_2 = \langle x_2, x_2', \sigma_2, \phi_2(V), \Phi_2(C), Reset_2(C), Assign_2(V) \rangle$ *are equivalent if* $\sigma_1 = \sigma_2$ *(same event),* $\phi_1(V) \equiv \phi_2(V)$ *(same equalities over the identical subset of variables in V),*$\Phi_1(C) \equiv \Phi_2(C)$ *(exact invariant condition on the clock variables),* $Reset_1(C) \equiv Reset_2(C)$ *(identical subset of clock variables is reset) and* $Assign_1(V) \equiv Assign_2(V)$ *(same subset of model variables with identical assignment). If* $\tau_1 \equiv \tau_2$ *then the source states of the transitions are equivalent and so are the destination states, i.e.,* $x_1 \equiv x_2$ *and* $x_1' \equiv x_2'$.
Source and destination states of an unmeasurable transition is measurement equivalent; as occurrence of unmeasurable transition cannot be identified so state change also cannot be detected.

Definition 2.2 (Projection operator): *A projection operator* $P : \Im^* \rightarrow \Im_m^*$ *can be defined in the following manner:* $P(\varepsilon) = \varepsilon$; $P(\tau) = \tau, \tau \in \Im_m$; $P(\tau) = \varepsilon, \tau \in \Im_{um}$; $P(s\tau) = P(s)P(\tau), s \in L_f(G), \tau \in \Im$, *where* ϵ *is the null string.*

The operator P erases the unmeasurable transitions from the finite argument trace. The term $P(s)$ is called *measurable finite trace* corresponding to the finite trace s.

Definition 2.3 (Measurement equivalent traces): *Two finite traces (or sequence of transitions)* s *and* s' *are measurement equivalent if* $P(s) = \langle \tau_1, \tau_2, ...\tau_n \rangle$, $P(s') = \langle \tau_1', \tau_2', ...\tau_n \rangle$ *and* $\tau_i E \tau_i'$, $1 \leq i \leq n$.

We use the symbol E to denote measurement equivalence of finite traces as well as that of transitions, with slight abuse of notation. The inverse projection operator $P^{-1} : \Im_m^* \rightarrow 2^{\Im^*}$ is defined as $P^{-1}(s) = \{s' \in L_f(G) | sEs'\}$ Thus, $P^{-1}(s)$ includes all possible sequence of transitions that are equivalent to the finite trace s. The projection operator P, the inverse projection operator P^{-1} and the notion of measurement equivalence E of finite traces can be extended to traces $\in \Im^w$, in a natural way.

2.4.2.2 Model with controllability

Definition 2.4 (Controllable Transition $\tau_c \in \Im_c$): *A transition $\tau_c \in \Im_c$ is said to be controllable if the corresponding event σ can be controlled by the system controller.*
In an uncontrollable transition $\tau_{uc} \in \Im_{uc}$ the corresponding event σ cannot be controlled by the system controller.

Given a state $x \in S$, with a single transition τ emanating from it, τ can be controllable or uncontrollable, depending on context. However, given a state $x \in S$, with more than one transition $\tau_1, \tau_2, \cdots, \tau_n$ say, emanating from it, if at least one transition τ_i, $1 \leq i \leq n$ is uncontrollable so are all the other transitions emanating from x. Let τ_1 be uncontrollable and τ_2 be controllable. As trigger of τ_1 cannot be disabled it may fire before τ_2 making it also uncontrollable.

Given a finite prefix trace $s = \langle \tau_1, \tau_2, \cdots, \tau_n \rangle$ of model G, s can be eliminated from $L(G)$ i.e., $L(G) = L(G) - s$ if at least one transition in $s = \tau_1, \tau_2, \cdots, \tau_n$ is controllable. If any transition τ_i in a trace s emanating from x is controllable then the event corresponding to τ_i can be disabled when the system is in state x thereby eliminating s from $L(G)$. The same holds for infinite traces.

2.4.2.3 Failure Modeling

Each state x is assigned a failure label by an unmeasurable status variable C with its domain $= \{N\} \cup 2^{\{F_1, F_2, \cdots, F_p\}}$, where $F_i, 1 \leq i \leq p$, stand for *permanent* failure status and N stands for normal status.

Definition 2.5 (Normal G-state): *A G-state x is normal if $x(C) = \{N\}$. The set of all normal states is denoted as X_N.*

Definition 2.6 (F_i-G-state): *A G-state x is failure state, or synonymously, an F_i-state, if $F_i \in x(C)$. The set of all F_i-states is denoted as X_{F_i}.*

Definition 2.7 (Normal-G-transition): *A G-transition $\langle x, x^+ \rangle$ is called a normal G-transition if $x, x^+ \in X_N$.*

Definition 2.8 (F_i-G-transition:) *A G-transition $\langle x, x^+ \rangle$ is called an F_i-G-transition if $x, x^+ \in X_{F_i}$.*

A transition $\langle x, x^+ \rangle$, where $x(C) \neq x^+(C)$, is called a *failure* transition indicating the first occurrence of some failure in the set $x^+(C) - x(C)$. Since failures are assumed to be *permanent*, there is no transition from any state in x_{F_i} to any state in x_N or from any state in $x_{F_iF_j}$ to any state in x_{F_i}. Also, for a transition $\langle x, x^+ \rangle$, $x(C) \neq \{N\} \Rightarrow x(C) \subseteq x^+(C)$. Failure transitions are unobservable and uncontrollable.

The following definition, proposed in [20], formalizes the notion of DES diagnosability of the failure F_i.

Definition 2.9 (**F_i-diagnosability**)*:*
Let $\Psi(X_{F_i}) = \{s \in L_f(G) |$ *the last transition of* s *is measurable and* $final(s) \in X_{F_i}\}$.
A DES model G *is said to be* F_i*-diagnosable for the failure* F_i *under a measurement limitation if the following holds*
$\exists n \in \mathbb{N}$ *s.t.* $[\forall s \in \Psi(X_{F_i})\{\forall t \in L_f(G)/s(|t| \geq n \Rightarrow D)\}]$,
where the condition D *is* $\forall u \in P^{-1}[P(st)], final(u) \in X_{F_i}$.

The above definition means the following. Let s be any finite prefix of a trace of G that ends in an F_i-state and let t be any sufficiently long continuation of s. Condition D then requires that every sequence of transitions, measurement equivalent with st (i.e., belonging to $P^{-1}(P(st))$), shall end into an F_i-state. This implies that, along every continuation t of s, one can detect the occurrence of failure corresponding to F_i within a finite delay, or more specifically, within at most n transitions after s.

Let there be two traces s_1 and s_2 that violate the F_i-definition of diagnosability; $P^{-1}(P(s_1)) = P^{-1}(P(s_2))$ and $P^{-1}(P(s_1))$ ends into an F_i-state while $P^{-1}(P(s_2))$ ends in into an non-F_i-state. If s_1 and s_2 are the only traces that violate diagnosability and s_2 be eliminated, then the model will become diagnosable.
The problem of active diagnosis ([21]): Given a DES model G *with controllable and uncontrollable transitions, eliminate minimum number of traces* S *from* $L(G)$ *such that* $L(G) - S$ *is diagnosable by Definition 2.9.*

2.4.2.4 The Diagnoser

In this subsection the construction method for diagnoser is given. The diagnoser is represented as a directed graph $O = \langle Z, A \rangle$, where Z is the set of diagnoser nodes, called *O-nodes*, and A is the set of diagnoser transitions, called *O-transitions*. Each O-node $z \in Z$ is

a set of G-states representing the uncertainty about the actual state and each O-transition $a \in A$ of the form $\langle z_i, z_f \rangle$ is a set of equivalent transitions representing the uncertainty about the actual measurable transition that occurs. Before discussing the procedure for constructing the diagnoser from G, the following definitions are introduced.

Definition 2.10 (Unmeasurable successor and unmeasurable reach of a set of G-states): *The* unmeasurable successor *(set) of a set Y of states is defined as $\mathcal{U}(Y) = \bigcup_{x \in Y} \{x^+ | \tau = \langle x, x^+ \rangle \in \mathfrak{I}_u\}$. The* unmeasurable reach *of a set Y of G-states, denoted as $\mathcal{U}^*(Y)$, is the reflexive-transitive closure of unmeasurable successors of Y.*

The diagnoser is constructed starting from the initial state(s) of the model. The states in X_0 are partitioned into equivalent subsets denoted as $X_{01}, X_{02}, \cdots X_{0m}$. For all i, $1 \leq i \leq m$, an initial O-node z_{0i} is obtained as the unmeasurable reach of X_{0i}, i.e., $z_{0i} = \mathcal{U}^*(X_{0i})$. The set of all initial O-nodes is denoted as $Z_0 = z_{01}, \cdots, z_{0m}$. The initial O-nodes capture the fact that the diagnoser can infer a set z_{0i} of the possible initial system states (or their unmeasurable reach) by measuring the variables without waiting for the first measurable transition. Given any O-node z, the O-transitions emanating from z are obtained as follows. Let $\mathfrak{I}_{mz}$ denote the set of measurable G-transitions from the states $x \in z$. Let A_z be the set of all equivalence classes of $\mathfrak{I}_{mz}$ under E. For each $a \in A_z$, a successor O-node z^+ of z such that $z^+ = final(a)$ can be created as follows. Let $z_a^+ = \{final(\tau) | \tau \in a\}$; then $z^+ = \mathcal{U}^*(z_a^+)$ and a is designated as: $\langle z, z^+ \rangle$. The set of the diagnoser transitions is augmented as: $A \leftarrow A \cup \{a\}$, and the set of O-nodes is augmented as: $Z \leftarrow Z \cup \{z^+\}$. Each $a \in A$ is an ordered pair $\langle z, z^+ \rangle$, where $z = initial(a)$ and $z^+ = final(a)$. Thus, each O-node contains equivalent states.

Henceforth, we refer to states, transitions and traces of G as G-states, G-transitions and G-traces, respectively. Similarly, for the diagnoser nodes and transitions, we use "O-nodes" and "O-transitions", respectively.

Definition 2.11 (F_i-O-node): *An O-node, which contains an F_i-state, is called an F_i-O-node, denoted as z_{F_i}.*

Definition 2.12 (F_i-certain O-node): *An F_i-O-node z is called an F_i-certain O-node if $z \subseteq X_{F_i}$. An F_i-O-node which is not F_i-certain is called* F_i-uncertain.

A path of the diagnoser O is a sequence of O-transitions $\gamma = \langle a_1, a_2, ... \rangle$, with consecution property. It may be noted that corresponding to any O-path $\gamma = \langle a_1, a_2, \cdots \rangle$, there is a

unique sequence of O-nodes $\langle z_1, z_2, \cdots \rangle$, where $z_i = initial(a_i)$ and $z_{i+1} = final(a_i)$, for $i \geq 1$. The term "O-path" is, therefore, used interchangeably for both O-transition sequence and O-node sequence. Similarly, "G-traces", is used interchangeably for both G-transition sequence and G-state sequence.

Definition 2.13 (F_i-uncertain cycle): *An* F_i-uncertain cycle *is an* F_i*-*O*-cycle in which there is no* F_i*-certain* O*-node.*

Definition 2.14 (F_i-indeterminate cycle): *An* F_i*-uncertain cycle* γ*, in which the* F_i*-states contained in the* O*-nodes of* γ *form a cycle in* G *comprising transitions from* γ*, is called an* F_i-indeterminate cycle.

The equivalence between F_i-diagnosability and the absence of F_i-indeterminate O-cycles has been formally established for DES models [20]. F_i-uncertain O-nodes comprise some normal G-states and some F_i-G-states. So, F_i-uncertain O-nodes cannot detect whether the system is normal or a fault has occurred. On the other hand, F_i-certain O-nodes can detect that a fault has occurred because the F_i-certain nodes comprise only F_i-G-states. Presence of F_i-indeterminate cycles imply that after failure F_i, the system can move indefinitely in F_i-uncertain O-nodes making the fault non-diagnosable. In active diagnosis, minimum number of traces from $L(G)$ are to be eliminated (if possible, because of controllability) such that there is no F_i-indeterminate cycles. In other words, in any F_i-indeterminate cycle γ, the F_i-state sequence (trace) contained in the O-nodes of γ which form a cycle in G comprising transitions from γ, is to be eliminated (if possible).

Now we present an abstract example to illustrate the active diagnosis problem. Figure 2.5 (a) illustrates active DES model of an abstract system. The model has seven states, namely a, b, c, a', b', c', d', where all the non-primed (primed) states represent normal (failure) behavior. In other words, $C(a) = C(b) = C(c) = N$ and $C(a') = C(b') = C(c') = C(d') = F_i$. The dotted transition ($failure$) represent occurrence of the failure F_i. We assume that transitions $1E1', 2E2'$ and $3E3'$, so states aEa', bEb' and cEc'. $failure$ is the only unobservable transition. Transitions $failure, 1, 2, 1', 2'$ are assumed uncontrollable, while $3, 3', 4'$ are controllable transitions.

The diagnoser for the DES model of Figure 2.5(a) is shown in Figure 2.5(b). Some of the initial steps diagnoser construction for this example are as follows.

(i) The initial state of the diagnoser i.e., z_1 is obtained as follows. X_0 is partitioned into

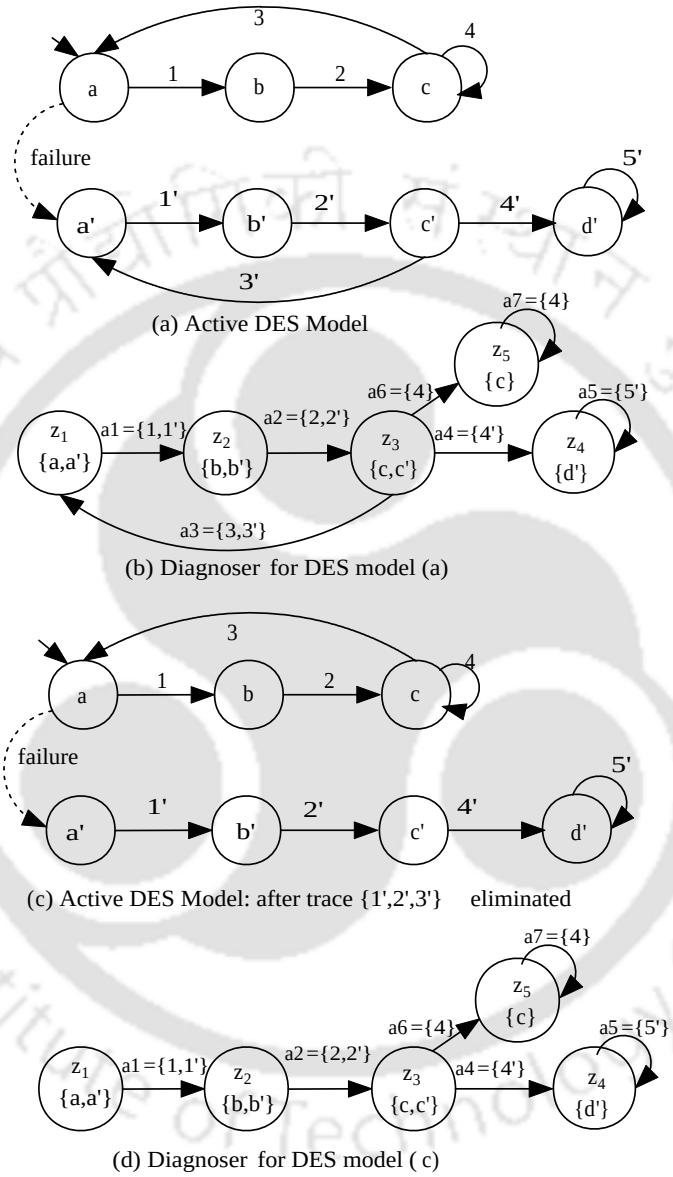

Figure 2.5: Active DES model for diagnosis of ARP

measurement equivalent subsets of G-states which in this case one i.e., $X_0 = a$; there is only one initial G-state a (as shown in Figure 2.5(a)) and $X_{01} = \{a\}$. As X_0 could be partitioned into only one subset of measurement equivalent initial G-states, $Z_1 = a$ and $z_{01} = \mathcal{U}^*(\{S_{01}\}) = \mathcal{U}^*(\{a\}) = \{a, a'\}$. Thus, there is only one initial O-node z_1 (as shown in Figure 2.5(b)) and $z_1 = \{a, a'\}$.

(ii) The outgoing O-transitions from z_1 are obtained as follows. Here, $\mathfrak{I}_{mz_1} = \{1, 1'\}$ which are all the outgoing measurable transitions from G-states in z_1. Now, $A_{z_1} = \{\{1, 1'\}\}$ as $1E1'$. Corresponding to $\{1, 1'\}$ there is an O-transition $a1$.

(iii) The destination O-node corresponding to $a1$ is obtained as follows. $z^+_{1_{a1}} = \{b, b'\}$ as $a1$ comprises G-transitions $1, 1'$ and $final(1) = b$ and $final(1') = b'$. Further, $z^+_1 = \{b, b'\}$ as $\mathcal{U}^*(\{b\}) = \{b, b'\}$ and $\mathcal{U}^*(\{b'\}) = \{b'\}$. Thus, the destination O-node of the O-transition $a1$ is $z_2 : \{b, b'\}$. Similarly, the construction of the diagnoser continues till no more new O-nodes are created in step (iii).

It may be noted that there is an F_i-indeterminate cycle $\langle z_1, z_2, z_3 \rangle = \gamma$ say (Figure 2.5(b)), making the fault non-diagnosable. Active diagnosis theory can be applied to see if the failure can be made diagnosable by eliminating some traces. In the Figure 2.5(b), F_i-state sequence (trace) contained in the O-nodes of γ (i.e., z_1, z_2, z_3) which form a cycle in G comprising transitions from γ (i.e., $a1, a2, a3$), is $\langle 1', 2', 3' \rangle$. As $3'$ is a controllable transition, its triggering event can be disabled when the system is in state c' (or c as cEc') which eliminates $\langle (1', 2', 3')^* \rangle$ from the language of the model; the model with this trace removed is shown in Figure 2.5(c) and the corresponding diagnoser in Figure 2.5(d). In the Figure 2.5(d), that there is no F_i-indeterminate cycle making F_i diagnosable. In other words, to make F_i diagnosable, when the system is in state c' (or c as cEc') the controller needs to enable event for transition $4'$ and the disable event for transition $4'$.

2.4.3 DES Modeling of ARP spoofing

Figure 2.6 illustrates the active DES model for ARP with probing under normal and request spoofing.

The values of different parameters in the active DES framework (Subsection 2.4.2) for modeling ARP is as follows.

$\Sigma = \{RQP, RSP, PRQP, PRSP, attack\}$. The set of states S is shown in Figure 2.6. States with

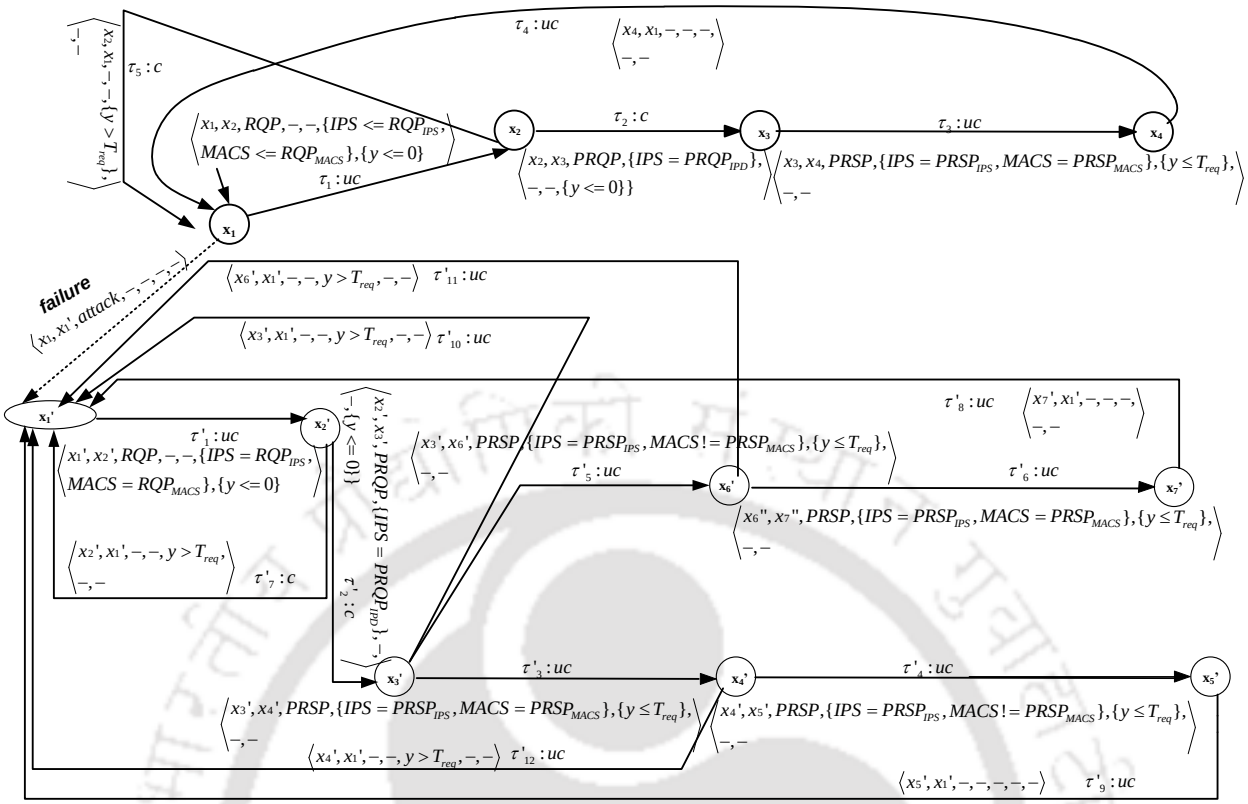

Figure 2.6: Active DES based model of ARP spoofing under normal and attack condition

no primes correspond to normal situation and those with single prime denote request spoofing. Initial state $X_0 = x_1$. Model variable set is $V = \{IPS, IPD\}$ and both its elements have the same domain given as $D_1(= D_2) = \{d.d.d.d | d \in \{1, 2, \cdots, 255\}\}$. There is one clock variable y. The transitions are shown in Figure 2.6; like states, transitions without primes are for normal model while single prime are for request spoofing. It may be noted that there are " − " for some fields in the tuple representing the transitions. If " − " is for $\phi(V)$ or $\Phi(C)$ then it represents TRUE condition, while if the " − " is for *Reset*(C) or *Assign*(V) then it represents NO action (i.e., reset or assignment) is to be taken. Controllable transitions are marked as u and uncontrollable ones by uc. The failure causing transition $x_1, x_1', attack, -, -, -, -$ (i.e., that cause request spoofing) is the only unmeasurable transition. The overview of the active DES model for ARP under normal and request spoofing cases are as follows.

Normal case

In the normal case (Figure 2.6), the model moves from x_1 to state x_2 on observation of an

event *RQP* by transition τ_1; this transition is uncontrollable, because any genuine host or attacker can send a response packet to another host. Enabling of τ_1 is only dependent on *RQP* and not on any model variables or clock invariant condition; this is depicted by " – " in the transition. Model variables *IPS* and *MACS* are assigned with source IP address and source MAC address of *RQP*, respectively. Clock variable y is reset to 0. Following that in state x_2, there are two options, either an ARP probe (*PRQP*) is sent (τ_2) or the system moves back to state x_1 after T_{req} (τ_5) time of receipt of the *RQP*. Transition τ_2 is enabled on *PRQP* (sent to source IP address of the *RQP* under question); correspondence of *PRQP* with *RQP* is determined by checking model variable *IPS* with $PRQP_{IPD}$. Also, clock variable y is reset. It may be noted that decision on sending the *PRQP* is made by the supervisory controller based on conditions like, (i) *PRQP* is sent if source IP-MAC of *RSP* is new (not yet verified), (ii) *PRQP* is not sent if extra traffic in network due to probing is to be minimized etc. So τ_2 is a controllable transition. If *PRQP* is not sent the system waits in state x_2 for T_{req} and then moves to x_1 via τ_5; clock invariant condition $y > T_{req}$ in τ_5 determines that T_{req} time has passed. As occurrence of τ_5 is dependent on τ_2 (which is controllable), so τ_5 also is controllable. After transition τ_2, probe response (*PRSP*) from the (normal) host arrives (τ_3); correspondence of *PSQP* with *RQP* is ensured by checking $IPS = PRSP_{IPS}$ and $MACS = PRSP_{MACS}$. Further, the *PRSP* is to arrive within T_{req} time after *PRQP* is sent; this is checked by the clock invariant condition $y \leq T_{req}$ in the transition τ_3. *PRSP*s are sent by hosts and are uncontrollable from the IDS perspective, making transition τ_3 uncontrollable. In the normal situation only one probe response would arrive for the probe request. So after receipt of one probe response the model returns back to the initial state x_1 by transition τ_4. The τ_4 fires instantaneously after the system reaches x_4, so there is no enabling event or condition. Due to absence of enabling condition τ_4 is an uncontrollable transition.

Request Spoofing case

The primed states and transitions in Figure 2.6 represent ARP request spoofing with probing by IDS. The model moves to state x'_2 on observation of an event *RQP* by transition τ'_1. Model variables *IPS* and *MACS* are assigned with source IP address and source MAC address of *RQP*, respectively. Clock variable y is reset to 0. Like τ_1, τ'_1 is an uncontrollable transition. Following that in state x'_2, there are two options, either an ARP probe (*PRQP*) is sent (τ'_2) or the system moves back to state x_1 after T_{req} (τ'_7) time of receipt of the *RQP*. As in the case of normal condition, τ'_2 and τ'_7 are controllable transitions. After transition τ'_2, there

are three options–(i) probe response from the attacker arrives (τ'_3) having $PRSP_{MACS}$ the same as spoofed RQP_{MACS} or (ii) probe response from the normal host arrives (τ'_5) having $PRSP_{MACS}$ not same as spoofed RQP_{MACS} or (iii) no response arrives (and the system moves back to state x'_1 by τ'_{10}) as it may be the case that source IP address of the request packet question RQP_{IPS} is non existent and the attacker does not sent any reply to the probe. As in the normal case, τ'_3, τ'_5 and τ'_{10} are uncontrollable transitions. Let transition τ'_5 fire and bring the system to x'_6. At x'_6there are two options–(i) probe response from the attacker arrives (τ'_6) having $PRSP_{MACS}$ the same as spoofed RQP_{MACS} or (ii) no other response to the probe arrives (and the system moves back to state x'_1 by τ'_{11}) as it may be the case that the attacker does not sent any reply to the probe. Transitions τ'_6 and τ'_{11} are uncontrollable. If τ'_6 occurs the system moves to x'_7 following which τ'_8 occurs instantaneously. In the attack situation more than one probe response would arrive for the probe request. So after receipt of more than one probe response the model returns back to the initial state x'_1 by transition τ'_8; τ'_8 is uncontrollable. In a similar way the other sequence of states from x'_3 (x'_4, x'_5, x'_1) can be explained. It may be noted that the model does not capture which probe response (*PRSP*) is from normal and which is from attacker. The model only denotes the fact that there are two responses with different MAC addresses; τ'_3, τ'_4 and τ'_5, τ'_6 are the two combinations of arrival of the responses with different MAC addresses. Further, these *PRSP*s are to arrive within T_{req} time after *PRQP* is sent.

2.4.3.1 Diagnoser for the DES model of ARP request spoofing

After development of the active DES model, we design a diagnoser by technique explained in Subsection 2.4.2.4 with a slight modification. In the diagnoser for the DES model of ARP request spoofing, no transitions and states are created from F_i-certain nodes. The diagnoser declares an attack when an F_i-certain node is reached as the estimate comprises states only from the attack model; no further estimation is required and one can stop at F_i-certain nodes. Figure 2.7 illustrates the diagnoser for the DES model of Figure 2.6.

In the diagnoser, nodes z_1, z_2, z_3, z_4 (z_5, z_6, z_7) are F_i-uncertain (F_i-certain) and attack cannot (can) be detected there. It may be noted that there are two F_i-indeterminate cycles. Repercussions of such indeterminate cycles on attack diagnosability and elimination of traces to break such cycles are as follows:

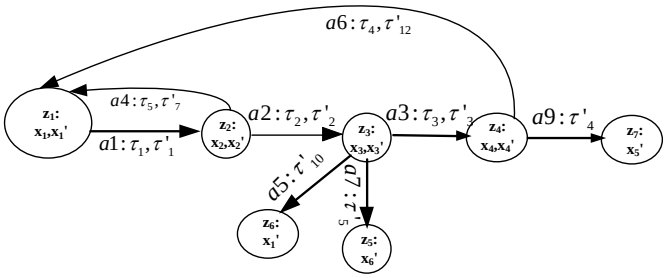

Figure 2.7: Diagnoser for the DES model of Figure 2.6

- $\langle z_1, z_2 \rangle$: This indeterminate cycle occurs if transition τ'_2 from x'_2 (τ_2 from x_2) is not fired, i.e., *PRQP* is not sent by controller. In other words, without sending a probe packet there is no difference in sequence of ARP events under spoofing attack and normal condition, thereby making attack (failure F_i) non-diagnosable. This indeterminate cycle can be broken by eliminating trace $\langle (\tau'_1, \tau'_7)^* \rangle$ (as τ'_7 and τ'_2 are controllable) which can be achieved by firing event *PRQP* at state x'_2 (or x_2).

- $\langle z_1, z_2, z_3, z_4 \rangle$: This indeterminate cycle occurs if transition τ'_4 from x'_4 is not fired, i.e., *PRSP* from the genuine host does not arrive. There can be several reasons when *PRSP* from the genuine host does not arrive namely: (i) Source IP address in *RSP* (i.e., $PRQP_{IPD}$) being verified is non existent. For that *PRQP* (τ'_2) with non-existing destination IP only the attacker responds (τ'_3) as only the attacker knows about the non existent spoofed IP address. (ii) Source IP address in *RSP* (i.e., $PRQP_{IPD}$) being verified is that of the attacker itself. $\langle z_1, z_2, z_3, z_4 \rangle$ can be broken by eliminating trace $\langle (\tau'_1, \tau'_2, \tau'_3, \tau'_{12})^* \rangle$. It may be noted that this is not possible because transitions τ'_4 and τ'_{12} are uncontrollable. So, if the source IP address in IP-MAC of *RQP* (τ_1 or τ'_1) is non existent or is that of a attacker, the spoofing cannot be detected. It may be noted that these two cases do not lead to severe situations. Associating IP address of non existent host or that of the attacker itself with different (false) MAC lead to diversion of traffic destined to non existent host or attacker (to the host that has the associated false MAC).

Now we discuss a case where the attack is detected. Let diagnoser reach node z_5 by the sequence z_1, z_2, z_3. z_5 being an F_i-certain O-node, reached by occurrence of $a7$ (by virtue of τ'_5), declares an attack. It may be observed from Figure 2.6 that τ'_5 corresponds to the *PRSP*

for a *PRQP* within T_{req} time whose source MAC is not same as the source MAC of the *RQP* under question.

2.5 Experimentation and Result

In this section we present the experimental testbed followed by results. The standard benchmarks for an IDS are accuracy, detection rate, resource consumption etc.In our case as some extra ARP messages are generated for probing, we also considered extra traffic statistics also as a benchmark along with accuracy, detection rate and resource consumption. The results in different situation of the network traffic is provided in this section.

In ARP request spoofing attack, an attacker sends (by broadcast) an ARP request packet with falsified IP-MAC pair (in sender hardware address-sender protocol address fields). The host with IP address given in the "target protocol address" field of the request packet updates its cache with wrong IP-MAC pairing. For example, let there be three hosts A,B and D; A and B are genuine and D is attacker. In request spoofing (targetting A) attacker D sends a forged ARP request packet with target protocol address as IP(A) and sender hardware address-sender protocol address as *IP*(*B*)-*MAC*(*D*); A will update its cache with MAC address of D corresponding to IP address of B. This will result in packets intended to be sent to B (by A) being sent to D. On the other hand if D sends a forged ARP request packet with *IP*(*B*)-*MAC*(*X*) (where *MAC*(*X*) is the MAC address of a non existing host) then all packets intended to be sent to B (by A) will be lost. ARP response spoofing is similar to ARP request spoofing. However, in the case of response spoofing, ARP response packet is used. For the sake of brevity, we will illustrate the proposed scheme for request spoofing attack only; the same will hold for response spoofing attacks also.

2.5.1 Testbed Architecture

As ARP is stateless, the sequence of ARP packets under normal and spoofed conditions is same. So for detection of spoofing, a mechanism is required that can create difference in packet sequence under normal and attack condition; this concept is used in the proposed IDS. The basic architecture of the network being monitored using the IDS is shown in Figure 2.8. As shown in the figure, the IDS comprises a DES diagnoser and a supervisory

controller. The supervisory controller (of IDS) receives all traffic due to port mirroring and sends probe requests to verify genuineness of ARP requests and responses; these correspond to controllable events. The fact that these probe requests create the required difference in packet sequence is illustrated using an example as follows. The diagnoser passively monitors all events namely, ARP requests, APR responses, probe responses etc. (uncontrollable events) and determines whether as attack has taken place.

An example scenario of spoofed response with probing:

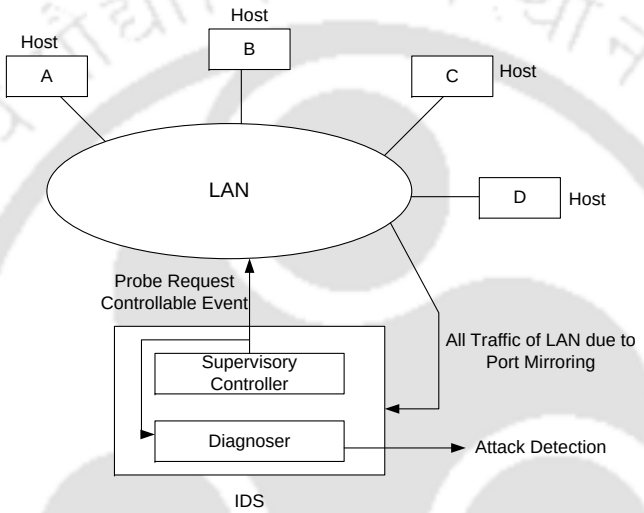

Figure 2.8: Basic Architecture of the testbed with IDS

In this example, the network has two genuine hosts A and B, D is the attacker and IDS is having IP=IP(E), MAC=MAC(E).

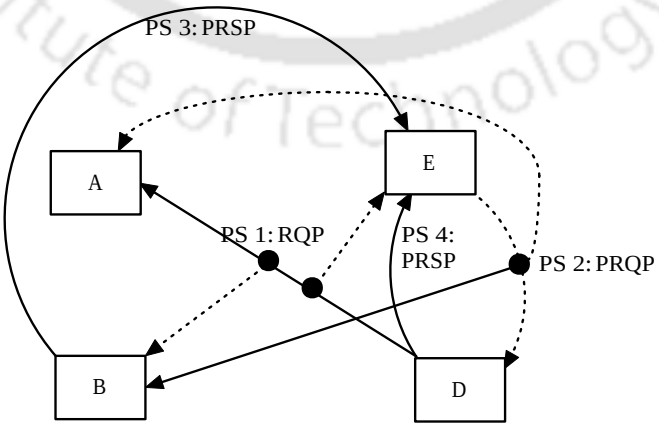

Figure 2.9: Example of Spoofed Request

Figure 2.9 shows the sequence of packets (indicated with packet sequence numbers) injected when attacker D is sending a spoofed request as "IP(B)-MAC(D) " to broadcast address asking host A's MAC address and its verification (using active probe). In packet sequence 1, D sends an ARP request *RQP* to broadcast address asking MAC address of A. After receiving this request message from D, A updates its cache with IP-MAC pair as IP(B)-MAC(D) first and then responds to D with its own MAC address. It may be noted that under normal condition "B's request to A with IP-MAC pair :IP(B)-MAC(B)" and attack condition "D's request to A with IP-MAC pair :IP(B)-MAC(D)", there is no change in sequence of ARP packets. However, under the attack condition, all traffic A wants to send to B will be sent to D.

Now use of active probing is discussed to highlight how it creates difference in sequence of ARP packets under normal and attack situations. On receiving the *RQP* from D to A (packet sequence 1), supervisory controller of IDS at E sends an ARP probe request *PRQP* to query the MAC address corresponding to IP(B) (packet sequence 2); *PRQP* is ARP request sent from IDS having source IP-MAC as IP(E)-MAC(E). As the ARP probe is a broadcast, both B and attacker D will receive the probe. B will definitely respond to the probe (*PRSP*) with IP(B)-MAC(B) to E (packet sequence 3) as it is assumed that non-comprised host (B) will always respond to the probe; this is an uncontrollable event. *PRSP* is an ARP response packet sent to IDS i.e., destination IP-MAC is IP(E)-MAC(E). Now the IDS can know that the request made in packet sequence 1 is false (as it had IP(B)-MAC(D)) and it can generate an alarm (and also track the attacker MAC (D)). To avoid self-identification, it is desirable for attacker D to respond to all queries (*PRSP*) for MAC address of IP(B), with IP(B)-MAC(D); this is an uncontrollable event as behavior of attacker cannot be assumed known. To summarize, at least one response to a probe (sent to verify IP(B), say) will arrive and have the genuine IP-MAC of B. In the case of spoofing, more than one responses will arrive which may have different MAC addresses.

In this example, it is assumed that the genuine host B responds (packet sequence 3) before attacker D (packet sequence 4) to the ARP probe from E. It may be noted that in the two responses, different MAC addresses are associated with the IP address being probed i.e., once MAC address of B is associated with IP address of B and then MAC address of D is associated with IP address of B. Table 2.5 lists the sequence of ARP packets for the example (shown in Figure 2.9).

Table 2.5: Packet and event sequence in the example of request spoofing

PS: Events	SRC IP	SRC MAC	Dest IP	Dest MAC
PS 1:*RQP*	IP(B)	MAC(D)	IP(A)	–
PS 2:*PRQP*	IP(E)	MAC(E)	IP(B)	–
PS 3:*PRSP*	IP(B)	MAC(B)	IP(E)	MAC(E)
PS 4:*PRSP*	IP(B)	MAC(D)	IP(E)	MAC(E)

Sequence of ARP traffic if the request packet (sequence 1) is genuine, (i.e., IP(B)-MAC(B) is sent to A by B) is given as follows. For the ARP probe sent by E to verify IP(B)-MAC(B) (packet sequence 2) only B will respond to E with IP(B)-MAC(B) (packet sequence 3). Till T_{req} time the IDS will receive only one *PRSP* (unlike two in case of attack). The IDS finds the response to be genuine (by virtue of only one *PRSP*).

2.5.2 Detection Rate, Accuracy and Resource Overhead

The metrics for determining efficiency of an IDS are accuracy, detection rate and resource consumption. In case of passive IDSs, resource consumption is in terms of memory and CPU utilization of the system executing the IDS. On the other hand, in case of active IDSs, as extra messages are generated due to probing, bandwidth utilization is another metric of resource consumption.

A test bed (as shown in Figure 2.8) has been created for experimental analysis of the proposed IDS. The testbed consists of 6 machines running different operating systems– Windows Xp, Fedora 16 (as router), Windows 7, Backtrack 5, Ubuntu and Windows 2008. Machine with Backtrack 5 is the attacker and machine with Ubuntu is configured as the IDS. These machines are connected using a CISCO catalyst 3560 G series switch with port mirroring enabled for the IDS.

The IDS is implemented in C++. For each ARP request or response packet received in the mirrored port, an event is generated by the supervisory controller, which in turn spawns an instance of the detector (Figure 2.6) at state s_1 and is added to an active list. On the arrival of the next event (i.e., *RQP*, *PRQP*, *PRSP*1, *PRSP*2), it is given to all instances of detectors in the active list to their corresponding present state. Each of them can make a transition on these events, if possible (based on the condition mentioned in Subsection

2.4.2.4). If at any (non-terminal) state, transition to a next state is not possible, attack is declared and the instance of the detector is deleted from the active list (thus releasing memory). Also, all detector instances reaching the terminal state are deleted. In other words, for detecting ARP spoofing, we create many instances of the detector machine. Each machine checks if the trace generated by the ARP event sequence belongs to the language generated by the diagnoser. By deploying attack generation tools Ettercap, Cain and Abel in host D and several scenarios of spoofing MAC addresses are attempted. Different amounts of attack packets (up to 2000) per second are injected in the test bed.

Table 2.6: ARP spoofing statistics

% of packets probed	no. of attacks launched	Detection rate (%)
100	2000	95.12
90	2000	89.76
80	2000	84.47
70	2000	79.08
60	2000	72.43
50	2000	67.91
40	2000	63.55
30	2000	60.36
20	2000	56.29
10	2000	52.57

Table 2.6 illustrates detection rate and accuracy versus percentage of ARP packets for which probe is sent. Obviously, with reduction in the number of probe packets sent, there is compromise in detection rate. Fall in detection rate with reduction in probing occurs because determining spoofed IP-MAC pair (which is not in Spoofed table) can be made only by sending a probe and receiving a response with non-matching MAC address. However, interestingly the rate of fall in detection rate is lower than the rate of reduction of probing. Specifically, with merely 10% of probe packets, the detection rate is still above 50%. The reason is, over a period of time Authenticated and Spoofed tables are built, where all the genuine and spoofed IP-MAC pairs are stored. These tables can detect attacks using the history and do not require sending of probes; this corresponds to event *DTD*. It may be noted that even for 100% probing, detection rate in not 100%, i.e., some false negatives are present. This is because of the cases like attacker spoofing itself etc., which cannot be detected. Among the different attack scenarios generated, 5% are the ones where the

attacker spoofs itself. This number is intentionally kept low, because the scenario where attacker spoofs itself does not have any impact.

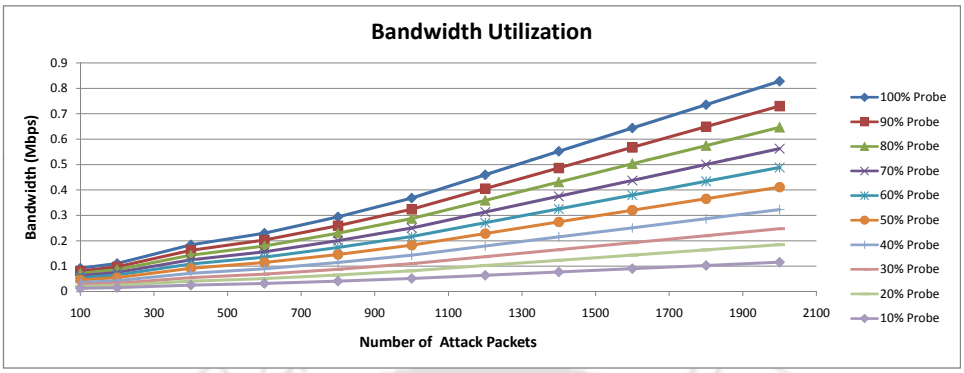

Figure 2.10: Bandwidth utilization at different percentage of probing

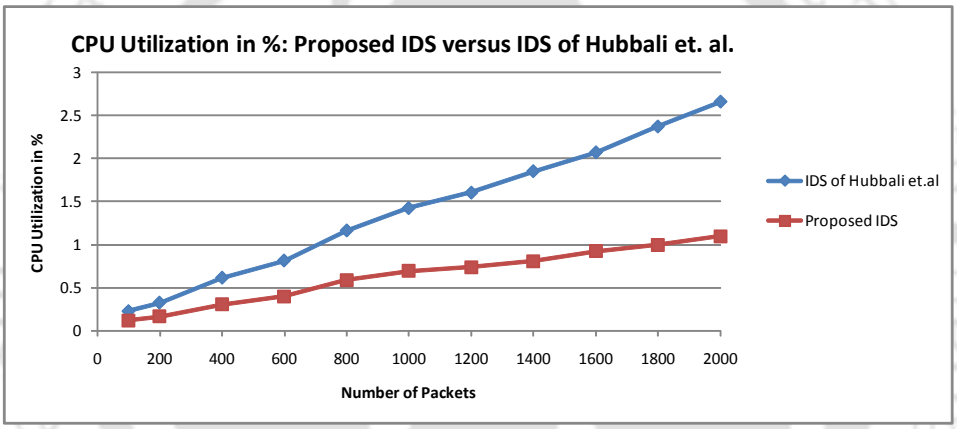

Figure 2.11: Comparison of CPU Utilization with IDS of Neminath et. al.

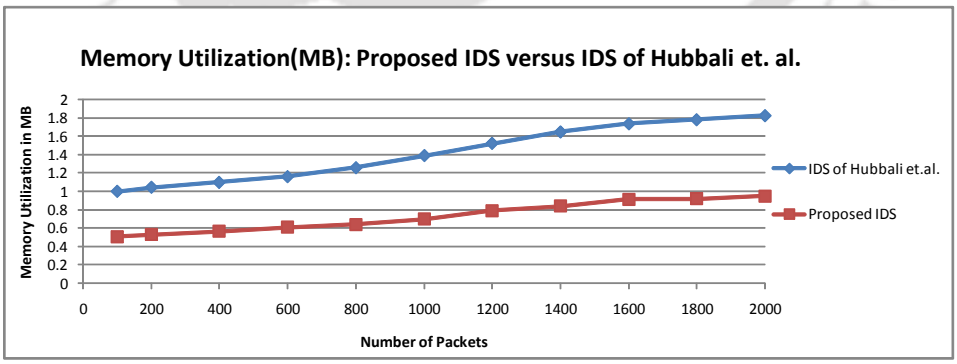

Figure 2.12: Comparison of Memory Utilization with IDS of Neminath et. al.

The accuracy is 100% (i.e., false positives are 0) under all cases because ARP spoofing is detected when (i) the IP-MAC pair being verified is found in spoofed table or (ii) for a probe,

a response arrives with non-matching MAC address. Case (i) comprises IP-MAC pairs already detected to be spoofed and Case (ii) can never happen under normal condition. So, the scheme never sends an alarm for a normal case, resulting in accuracy of 100%.

Figure 2.10 shows bandwidth utilization due to ARP traffic under different percentages of probing. As probes generate extra ARP traffic (due to probe itself and replies), bandwidth decreases with reduction in percentage of probing.

Figure 2.11 and Figure 2.12 (red line) show CPU and memory utilization (at 100% probing), respectively when running the IDS in Intel Core-2-Duo host with 2 GB RAM. In these cases also, CPU utilization and memory utilization fall with reduction in percentage of probing; as the trends are similar to that of bandwidth, we do not present them explicitly.

Figure 2.11 and Figure 2.12 also compare CPU and memory utilization of the proposed IDS with that of [17]. As many instances of the detector may be spawned, memory and CPU utilization of the host executing the IDS is in tolerable limit and does not over burden the host. This fact is illustrated in Figure 2.11 and Figure 2.12.

Bandwidth utilization of both [17] and the proposed IDS is similar. This is because both the IDSs are active thereby sending probes for IP-MAC pairs whose genuineness is not known. For each probe there is one or two responses based on attack or normal condition. So extra ARP traffic is similar for both the IDSs i.e., similar bandwidth requirement.

Regarding resource overhead of the proposed scheme, it may be noted that CPU utilization, memory utilization and bandwidth utilization, even at 100% probing, is minimal compared to resource available in modern infrastructure.

2.6 Conclusion

ARP spoofing is a major attack using which other attacks like MiTM, DoS etc. can be launched. In this chapter we proposed an active mechanism to detect ARP based attack using active DES based IDS. The scheme uses an active probing mechanism based on ARP requests and responses, so it does not violate the principles of network layering architecture. Further, this being a software based approach, does not require any additional hardware or software patching in the hosts. The scheme being based on formal state-transition modeling, it is verifiable.

This active DES technique is useful in detecting attacks where normal and attack traces

can always be differentiated using network events (ARP requests probes in this case) which are controllable. However, there are certain attacks where some of the network events required to distinguish normal traces from attack traces may be uncontrollable. For example, in ICMP based attacks, sending of probes and receiving their responses depend on congestion in the network. As congestion is an uncontrollable event, so sometimes traces under attack condition cannot be differentiated from the normal condition. This leads to the DES detector to reach some attack/normal uncertain state.

This motivated us to explore the possibilities of detecting ICMP based attacks by diagnosing only those traces where the detector leads to some attack/normal certain states, leaving out the paths where it is uncertain; this is termed as partial diagnosis. In the next chapter we propose a partial DES diagnosability approach for developing an IDS for ICMP attacks.

Chapter 3

Partial Diagnosis of DES based detection for ICMP related attacks

3.1 Introduction

Internet Protocol (IP) is the primary protocol in the network layer of the TCP/IP protocol suite and has the task of delivering packets from the source host to the destination host solely based on their network addresses. The lack of reliability in this protocol allows many of the fault events to occur like data corruption, packet duplication, out of order delivery. Therefore, some mechanism is needed to control and notify the errors in the network layer. Internet Control Message Protocol (ICMP) is used by the IP to send one-way informational messages to implement various error-reporting, feedback and testing capabilities.

There is no authentication mechanism in ICMP, which leads to attacks using ICMP that can result in a Denial of Service(DoS) or allowing the attacker to intercept packets. Some of these ICMP messages can cause a host to drop or reset a connection immediately. An attacker can simply forge one of the ICMP ”error messages” and send it to one or both of the communicating hosts leading to connection breaking. If an attacker forges an ICMP Redirect message, the attacker can siphon packets intended for another host. By using ICMP ”informational messages”, attacker can launch different attacks like ping of death, ping flood and smurf. ICMP tunneling, port scanning and OS fingerprinting are some of the other attacks possible using ICMP messages.

For these ICMP based attacks, writing a signature is not possible as they that do

not change the syntax and sequence of network traffic under normal and compromised situations. So, it is not possible to detect these attacks using signature based IDS. Further these attacks do not lead to any significant deviation in network traffic characteristics. So, if anomaly based IDSs are used to detect these attacks, false alarms are extremely high.

As discussed in the last chapter, active DES technique is useful in detecting attacks for which signature and anomaly based IDSs are not efficient. However, for successful diagnosability in active DES framework it is mandatory that normal and fault traces should always be differentiable using events which are controllable. It may be noted that in case of ARP, normal and attack traces can always be differentiated using network events (ARP requests probes in this case) which are controllable. However, in ICMP based attacks, sending of probes and receiving their responses depend on congestion in the network. As congestion is an uncontrollable event, so sometimes traces under attack condition cannot be differentiated from the normal condition. This leads to a condition where the DES detector reaches some attack/normal uncertain state. So active DES framework cannot be applied for ICMP attacks.

In [20] authors have proposed a framework called I-diagnosability, for partial diagnosis problems. Here some indicator events are defined and failure diagnosis is tested only in those paths where a fault is followed by an indicator event. The paths where there are no indicator events, may contain some fault/normal uncertain states, but that does hinder diagnosis of the system as a whole. For ICMP related attacks, the paths where there is issue like congestion, no indicator event is available. Thus, this restricted diagnosability framework [20] is adopted for modeling and designing IDS for ICMP related attacks. Detector that is designed using this framework is called *I*-detector. It is also found that the *I*-detector constructed following the method proposed by Meera et al. [20] contains many redundant states that do not help in diagnosis purpose. So a reduced *I*-detector (called *RI*-detector) is also proposed in this chapter. Construction and runtime complexity of this *RI*-detector is less than *I*-detector.

In this chapter, various security issues and attacks related to ICMP is described. Among various ICMP attacks, Network/Host Unreachable messages based attack is considered and an IDS is designed using I-diagnosability based DES framework. The detection scheme is successfully validated in a testbed with various attack scenarios and the results show the effectiveness of the proposed technique. The proposed approach does not require

any ICMP packet format modification or modification of any network infrastructure. It also ensures that the layering architecture is not violated. As the detection mechanism is centralised, it does not require patching in all machines in the network. It requires very less memory and CPU resources. Being this is a software based approach, it does not require any additional hardware.

The rest of the chapter is organized as follows. In Section 3.2 the ICMP attacks and proposed schemes in literature to defend against these attacks are discussed. The motivation for using the I-diagnosability based DES framework for attack detection is also noted here. Section 3.3 describes the proposed DES based approach to combat ICMP Network and Host unreachable attacks. Following that, in Section 3.4 the algorithm of *RI*-detector construction is proposed and it is shown that it runs in lesser time complexity than *I*-detector. Finally the chapter is concluded in Section 3.6.

3.2 ICMP based attacks and existing schemes to mitigate it

3.2.1 Attacks using ICMP Messages

ICMP is used by the IP layer to send one-way error and informational messages to a host. There is no authentication mechanism in ICMP, which leads to attacks using ICMP that can result in a DoS, or allowing the attacker to intercept packets. The detailed description of each attack and how they are launched using various ICMP messages are now discussed.

3.2.1.1 Destination Unreachable

Destination Unreachable messages [22] are used to inform the host that destination is unreachable due to some reason. These errors may be generated as a result of TCP, UDP or another ICMP transmission. Destination Unreachable messages can be generated by either the host or the gateway. Messages like Net unreachable, Host unreachable, Fragmentation needed but DF set, Source Route failed etc. are sent by gateways and Port Unreachable, Protocol Unreachable etc. are sent by hosts.

From Gateway

Net Unreachable normally occurs when the destination network is unreachable or down. Usually gateway generates this error when the entry for destination network is not there in routing table. This error message can be spoofed by the attacker and can be used to stop a host connecting to the destination network which results in DoS.

Host Unreachable is generated by the boundary gateway or router of different subnet. It is send by router when it receives a datagram which it cannot deliver or forward to destination host. An attacker can spoof this error message to stop the source from communicating with the destination host.

Fragmentation needed but DF set error is generated by gateways if the size of datagram is very large and is needed to be fragmented but in IP header "Don't Fragment" bit is set. This kind of message is used in Maximum Transmission Unit (MTU) discovery mechanism [49] in which the host sends IP packets with DF flag set in IP header to determine the MTU size. On receiving this error message it decrements the size of the IP datagram. These kind of messages can be spoofed to stop host from sending the datagram telling that messages should be fragmented. Another attack using this Destination Unreachable message is described in [50] which is known as the Performance-Degradation attack. In this attack, the attacker sends this error message with a small value in the Next hop MTU field notifying the host that the size of datagram should be lesser than specified MTU. In such a case, the host becomes busy in fragmentation, which degrades its performance and eventually leads to DoS.

Source route Failed message is generated by gateways when the route specified in IP options cannot be followed due to some reasons or the datagram cannot be forwarded to next hop. The attacker can spoof such messages to fool the host that IP datagram cannot follow the specified route.

From Hosts

Protocol Unreachable occurs when the designated transport protocol (addressed in the IP protocol field) is not supported at the receiving host. It allows the source to know whether a particular IP protocol (TCP, UDP, ICMP, IGMP etc.) is supported by the target machine or not.

Port Unreachable occurs when a source is trying to connect with destination through a port which is closed or access rule to the port is violated. If IP is carrying a higher layer protocol such as TCP or UDP, it is possible that the destination host is not configured to support that transport protocol. In such case, the destination host returns this error message if it has no protocol mechanism to inform the sender.

Attacker can spoof the Protocol Unreachable and Port Unreachable messages and can abort the ongoing connection between two hosts. This is referred to as the Connection-Reset Attack in [50].

3.2.1.2 Source Quench

Source Quench messages [22] are generated by destination host when it receives datagram at a rate that is too fast to be processed. Gateway reports this error message if it does not have enough buffer space to queue the datagrams to forward to the next-hop. The source quench message is a request to the host to cut down the rate at which it is sending traffic to the destination. Attacker can use forged Source Quench messages to slow down a host from responding, which can lead to Throughput-Reduction attack [50].

3.2.1.3 ICMP Redirect

ICMP Redirect messages [22] are used by routers to notify the hosts that a better route is available to reach a particular destination. ICMP redirects can hence be used by the attackers to cause false re-routing of packets. An ICMP redirect attack sends ICMP redirect messages to hosts on a subnet to request the hosts to change their routing tables. These ICMP messages can be exploited by the attacker to set up a false route to destination and apply a Man-in-the-Middle attack.

3.2.1.4 ICMP Router Discovery Messages

These messages allow the hosts to find out the IP addresses of the router that is attached to their immediate network. Since these messages are not authenticated, attackers on the same subnet can spoof these messages leading to Passive Monitoring, MiTM, DoS etc. [51].

3.2.1.5 ICMP Echo request and reply

ICMP Echo request and reply messages [22] are mainly used for connection testing. When a host wants to check whether it can reach a destination or not, it begins the test process by sending an Echo request message to the destination host. If the destination responds back with an Echo reply, the source ascertains that destination is reachable [52].

ICMP echo request/reply messages are mainly used by the attackers for *reconnaissance* and *scanning* purposes. Another serious attack is possible by flooding the target host with huge amount of ICMP echo messages to launch DoS attack. *Smurf attack* [53] is a good example one such attack, where the attacker broadcasts ICMP echo request packets to whole subnet by spoofing the victim's IP address as the source address. All the hosts on that network will receive such an ICMP echo request and hence respond with an ICMP echo reply to the victim. This leads to DoS of the victim and can also cause network congestion.

3.2.2 Existing Schemes for Detecting ICMP Attacks

Different detection mechanisms available for identifying ICMP related attacks are discussed in the following subsections.

3.2.2.1 Signature-based Detection Systems

As a rule-based IDS, Snort [4] uses signatures to identify some types of ICMP attacks such as smurf, ping of death and network scanning. Based on these rulesets, packets are dropped when the signatures are matched.

Cisco Firewall [23] has a set of signatures to protect the internal network. For instance, it checks if ICMP packets have "More Fragments" flag set to 1 and whether the length in IP header is > 1024 bytes. Likewise, another signature for protection against DoS in Cisco routers limits the number of ICMP unreachable packets transmitted to one pkt per 500 ms.

The whitepaper [24] suggests to block/drop the packets related to ICMP redirect and ICMP unreachable messages to avoid DoS attacks. For Port Scanning, it checks the number of TCP/UDP packets and collects statistics on how many destination ports, a source IP is sending such packets. To avoid ICMP flood, it checks the rate of ICMP packets destined for specific destination address.

The main problem with the signature-based detection systems is that they have predefined rules or signatures only for a particular group of attacks. Attacks with slight variations would not be detected by these IDSs.

3.2.2.2 Router Configurations

Default router configurations are defined for protecting networks. Such configurations allow ping (ICMP Echo request outbound; Echo reply inbound); allow traceroute (TTL Exceeded and port unreachable inbound) and allow PathMTU (ICMP Fragmentation needed but DF set inbound). Any ICMP packet not satisfying the specified configuration is blindly dropped by the routers.

In [25], Chau suggests the router configuration to block all the outbound packets that indicate a source address which is not a part of the subnet. Also, the router should disallow any broadcast packets i.e., it should be configured to block all network prefix directed broadcast packets.

These router configurations block much of the essential ICMP packets that may not be attack packets, which is not desirable. Every ICMP message has its own significance and hence they should not be dropped blindly.

3.2.2.3 Extensions to ICMP Messages

In [26], ICMP traceback messages are used to learn the path that packets take through the Internet. This is especially important for dealing with certain DoS attacks, where source IP is forged. When forwarding packets, routers can generate ICMP Traceback messages. With enough traceback messages, source and path of forged packets can be determined. But such extensions to ICMP messages require changes to the global routing infrastructure which is not practical.

3.2.2.4 Cryptographic Techniques

In [27], secure mechanism for router discovery for hosts in IPv6 is presented. IPv6 nodes use neighbor discovery protocol to discover other nodes on link to determine their link layer address to find routers and to maintain reachability information about path to active neighbors. In [40], SEND introduces Cryptographically Generated Address (CGA). There

are chances that proposals for using such cryptographic techniques to add authenticity to the ICMP messages may be used. But these have costly computational overhead and high storage resource requirement. Moreover, implementing such cryptographic algorithms on global routers will be a real challenge.

From the review, it may be stated that an ICMP attack prevention/detection scheme needs to have the following features

1. Should not modify the standard ICMP protocol.
2. Should generate minimal extra traffic in the network.
3. Should not require patching, installation of extra software in all systems/routers.
4. Hardware cost of the scheme should not be high.
5. Should provide a more formal and intuitive framework for the attack diagnosis.
6. The complexity of attack diagnosis should be low to make it work for large interconnected networks.

This chapter exhibits use of I-diagnosability framework for detecting ICMP NHU attack. First, DES framework [21] is applied to design an IDS for the attack, however, it is found to be non-diagnosable. The reason was the presence of some normal/fault uncertain traces in the DES detector. To address this issue, I-diagnosability [20] paradigm is applied to perform diagnosis only in paths where fault is followed by some indicator event. The attack is found to be I-diagnosable and finally an *I*-detector is constructed. In order to reduce the complexity of the *I*-detector, an algorithm for a reduced I-detector, called *RI*-detector, is proposed. The proposed *RI*-detector meets all the above mentioned requirements. To best of the knowledge there is no previous work reported on modeling of IDSs with *RI*-detector. Advantages of this attack detection technique over the previous ones are:

- The proposed technique requires the IDS to run on a single host instead of modifying every hosts/routers of the network.
- It does not violate the standard ICMP protocol and follows the principle of network layering structure.
- The *RI*-detector runs in much lesser complexity so the overhead gets reduced.

In the next section the concept of diagnosis of ICMP NHU attack is explained, the detector is constructed and it is found to be partially diagnosable.

3.3 IDS for ICMP Network/Host unreachable (NHU) attack using the Failure Detection Theory of DES

A DES, as already mentioned is characterized by a discrete state space and some event driven dynamics. The basic idea is to develop a DES model for the system under normal condition and also under each of the attack conditions. Following that, a state estimator i.e. the detector is designed which observes sequences of events generated by the system to decide whether the states through which the system traverses correspond to the normal or faulty DES model. Note that an attack condition is treated as a fault here, as stated earlier. Thus, for detection of NHU attack, first the system has to be modeled as a DES. For that, the network architecture on which the scheme is to be validated is presented. The active probing mechanism is discussed thereafter.

3.3.1 Network Architecture

The network architecture shown in Figure 3.1 is used to illustrate the scheme. Hosts A, B, C and D are connected to router R1. The IDS monitors the network traffic within its LAN through its mirrored port. Port mirroring is enabled at the switch so that the IDS C has a copy of all the outgoing and incoming packets from all ports. Routers R2, R3 and R4 have their own networks with hosts X, Y and Z respectively. NetFlow monitoring [54] is enabled in all the interfaces on the first hop routers which are directly connected to the IDS. Hence, the IDS monitors those ICMP packets received by the outer interfaces of R1 (i.e. ICMP traffic from R2 and R3 towards R1) as well. The network assumptions are noted as-

1. All the systems which are being monitored are up; and running and they reply to the IP probes within a specific threshold interval t_{icmp}.
2. Port mirroring is enabled at the switch in the local subnet of the IDS.
3. The system administrator is aware of the routers (IPs) directly connected to the network and he maintains a whitelist of such IPs as $IP_{GATEWAY}$.

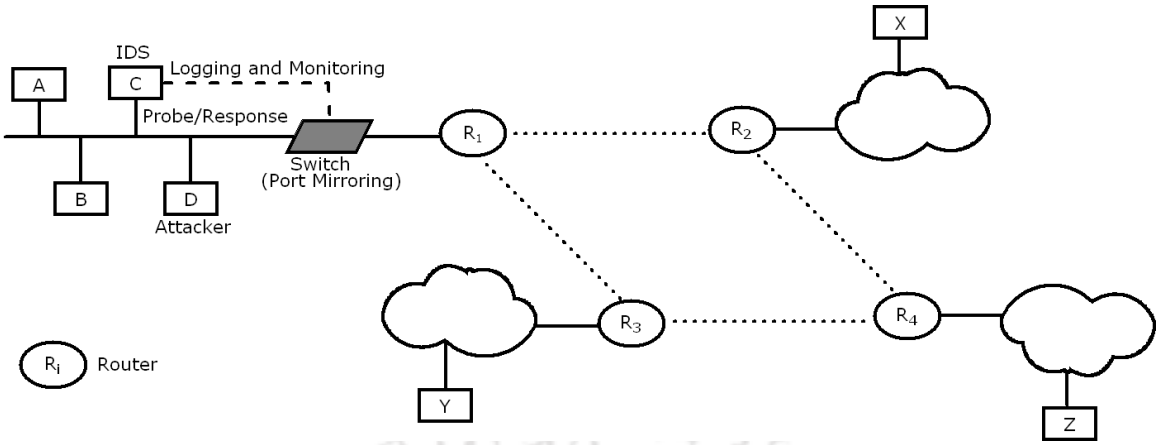

Figure 3.1: DES detector for NHU attack

4. SNMP with authentication support [55] is enabled in all the routers which are directly connected to network. Hence, spoofing of SNMP messages is not possible.

As the spoofed packets in ICMP attacks looks similar to the corresponding packets generated under normal conditions, an active probing technique is used so that sequence of ICMP packets can be differentiated under normal and attack conditions. Inside the network, the IDS takes care of this probing. The proposed scheme relies on the assumption that attacker can not stop the genuine host from sending a reply to the probes initiated by the IDS. IDS has two modules running inside it,

- **Handlers:** The handlers send probe packets when it is required and intimates encountered events to the other module i.e., the detector module. By virtue of port mirroring IDS gets all the ICMP packets generated in the network. Depending on the type of ICMP message encountered, one of the handlers is invoked. After being invoked, the handler processes the packets and generates appropriate events. The handlers will be explained in the next subsection.

- **Detector:** The detector module which is obtained as result of the failure detection procedure, runs in background inside IDS. It observes the event sequence given to it and decides whether an observed sequence corresponds to attack condition or normal condition.

3.3.2 Active Probing Technique

The active probing technique is explained here with respect to an IDS for *ICMP Net/HostUnreachable* attack. The same methodology has been applied for the other attacks as well. As already discussed earlier, in *Network Unreachable* attack, an attacker spoofs as a router and sends a false ICMP Net Unreachable packet to prevent the victim host from connecting to the destination network. Similarly for the *Host Unreachable* attack, an attacker can spoof a ICMP Host Unreachable packet and thus prevent the victim host from connecting to its destination host. Upon receiving ICMP Net / Host Unreachable packets, *ICMP Echo Probes* are send for attack detection. To assist in the probing and separating the genuine ICMP messages with that of spoofed ones, the following tables are maintained.
The following short notations are used hereafter: *IPS* - Source IP Address, *IPD* - Destination IP Address, *NHU* - Net/Host Unreachable, *NU* - Net Unreachable.

1. *SNMP_TABLE*: A SNMP query is generated for retrieving various information from the router. Such SNMP queries are initiated by the IDS and are called as SNMP probes. The SNMP probe sent, requests for the values of the Object Identifiers(OID) specified in OID_LIST. The OID_LIST is a tuple which contains a single OID or a set of OIDs depending on the requirements of the SNMP query. The values corresponding to each of the OIDs in the OID_LIST is returned in the tuple denoted by OID_VALUE.

   On receiving the SNMP packet, the details are entered in the *SNMP_TABLE*. The destination IP address, OID_LIST, OID_VALUE and the timestamp are stored in this table. Each entry in the *SNMP_TABLE* is represented by *SNMPT*. This table has four fields as Destination IP Address ($SNMPT_{IPD}$), list of Object Identifiers OID_List ($SNMPT_{OID\_LIST}$), list of values OID_Value ($SNMPT_{OID\_VALUE}$) and Timestamp ($SNMPT_{\tau}$).

2. *LOG_TABLE*: Whenever an attack is detected, the type, code, source IP address and destination IP address of the spoofed packet is stored in this table along with the time stamp referring to when the attempt is detected. It is a history of attack attempts and acts like a log file. The table has five fields and is denoted by *LOGT*. Its fields are type ($LOGT_{type}$), code ($LOGT_{code}$), Source IP Address ($LOGT_{IPS}$), Destination IP Address ($LOGT_{IPD}$) and Timestamp ($LOGT_{\tau}$).

3. *REDIRECT_table*: The fields are IPS ($REDIRECTT_{IPS}$), Preferred Gateway ($REDIRECTT_{prefgw}$), code ($REDIRECTT_{code}$), IPD ($REDIRECTT_{IPD}$), $REDIRECTT_{timeofsending}$, $REDIRECTT_{count}$, $REDIRECTT_{TTL}$, $REDIRECTT_{count}$ and $REDIRECTT_{TTL}$ are updated later when response to probe arrives.

   $\langle TableName \rangle_{MAX}$ represents the maximum number of elements in a table at a given time.

3.3.2.1 Proposed Algorithms

The IDS, as already mentioned in the assumptions, gets a copy of all the incoming and outgoing packets from all ports as port mirroring is enabled. Some initial checks are performed on these in order to optimize the handler algorithms. On identifying *ICMP* packets, checks are performed to find out whether they are malformed or not. If they are malformed, an alarm is generated and they are dropped right away even before invoking the respective handlers.

Also, at this point, it is possible to check for the ICMP packets' correspondence to some of the pre-existing signatures for ICMP attacks like *ping of death* and *smurf*. In case of ping of death, the sum of the IP offset and data length would be greater than 65535 bytes. If this signature is matched, the attack is notified to the system administrator immediately. ICMP Echo messages are checked for the signature of the smurf attack. The smurf attack aims at broadcasting ICMP Echo requests with the spoofed IP of the victim as the source address. Hence, if the destination IP is found to be the broadcast domain 255.255.255.255, smurf can be detected. Similarly, other signatures for prominent attacks can be checked at this stage in order to avoid much processing time.

For the sake of optimization, there are two global variables which are used through out all the described algorithms – *CONGESTION_STATE* and *ICMP_COUNTER*.

1. *CONGESTION_STATE* is used to denote the state of Congestion Check module. *CONGESTION_STATE* can be *True*, *False* or *Unspecified*.
   CONGESTION_STATE is set to be *Unspecified* if Congestion Check module has not been called for *t_congestion* time. Before the Congestion check module is called, the *CONGESTION_STATE* is first checked. Only if its state is *Unspecified*, the Congestion Check module is called. The *CONGESTION_STATE* is set as *True* or *False* depending

on the results returned by the Congestion Check module. Otherwise, it directly utilizes the value of the *CONGESTION_STATE* variable. This saves a substantial amount of processing overhead.

2. Also, in order to avoid flooding attack, the IDS maintains *ICMP_COUNTER* which can be incremented on the reception of ICMP messages. When this counter value reaches threshold N_{ICMP} within t_{flood} time, an anomaly is detected due to the sudden increase in the number of packets and an IDS ALERT signal is generated. On receiving this signal, the IDS gets into the alert mode and stops the processing of flooding packets. When the packet rate gets back to normal, the usual processing all ICMP packets of the IDS continues.

Now the algorithms for detecting ICMP NHU attack is discussed. The other algorithms used to detect rest of the ICMP attacks will be discussed later. The probing technique used for detection of NHU attack uses four main handlers namely, *CONGCHECK*(), *NHUHAND*(), *ECHOREPLYHAND*() and *EXPHAND*(). These are elaborated in Algorithm 3.1, Algorithm 3.2, Algorithm 3.3 and Algorithm 3.4 respectively. Algorithm 3.1 describes *CONGCHECK*() i.e. the congestion check module used to detect congestion in the gateway. *CIP*, *CTYPE* and *CCODE* are used to identify the *ICMP* packet encountered and for which the *CONGCHECK*() is invoked. It uses SNMP queries to retrieve the congestion related information. High bandwidth utilization attributes to congestion. Hence, the presence of congestion could be inferred from an estimate of the interface's bandwidth utilization [56]. Bandwidth utilization is based on three parameters – *ifInOctets* - Number of Octets input to the interface , *ifOutOctets* - Number of octets coming out from the interface and *ifSpeed* - Interface speed at that moment. Therefore, the SNMP probes are sent for the OID_LIST tuple ($SNMP_{OID\_LIST}$) with three fields
$< ifInOctets, ifOutOctets, ifSpeed >$.
The three fields in the OID_VALUE tuple ($SNMP_{OID\_VALUE}$) i.e.
$< OID\_VALUE1, OID\_VALUE2, OID\_VALUE3 >$ are initialized to NULL. The values corresponding to the three OIDs are returned in the OID_VALUE tuple i.e.
$SNMP_{OID\_VALUE}$.

Algorithm 3.1 explains the steps involved in the congestion detection mechanism. Algorithm 3.1 takes *CIP* (IP address of gateway to be tested for congestion), *CTYPE* (ICMP

message type), *CCODE* (ICMP message code) as parameter. To measure the bandwidth utilization, take two set of values of ifInoctets and ifOutOctets at a time interval of t_{snmp} seconds. These values are specified as $ifInOctets_{old}$, $ifInOctets_{new}$, $ifOutOctets_{old}$ and $ifOutOctets_{new}$. These values are then substituted in the formula for bandwidth utilization estimation [56] and the result *Bandwidth _Utilization* is computed. Now if *Bandwidth_Utilization* is more than the specified $Threshold_{Bandwidth}$, *CONGESTION_STATE* is returned as *True*; otherwise, it is *False*. The events *CON* and *NOCON* are intimated accordingly. Also, *CTYPE* and *CCODE* are intimated. This congestion check is usually performed for the interface directly connected to the subnet.

Algorithm 3.1 *CONGCHECK*(**CIP**,**CTYPE**,**CCODE**)

Input : *IP* - IP address of host, *type* of ICMP packet, *code* of ICMP packet, *SNMP_TABLE*, *CONGESTION_STATE*

Output: Intimate congestion in network as *CON* and no congestion as *NOCON*

Send SNMP Query to *CIP* for $SNMP_{OID\_LIST}$;
On receiving the SNMP response, store the values in $< VAL1, VAL2, VAL3 >$ corresponding to $SNMP_{OID\_VALUE}$;
Add *CIP*, $SNMP_{OID\_LIST}$, $SNMP_{OID\_VALUE}$ and τ to *SNMP_TABLE*;
$ifInOctets_{old} = VAL1$;
$ifOutOctets_{old} = VAL2$;
Wait for t_{snmp} seconds;
Send SNMP Query to *CIP* for $SNMP_{OID\_LIST}$;
On receiving the SNMP response, store the values in $< VAL1, VAL2, VAL3 >$ corresponding to $SNMP_{OID\_VALUE}$;
Add *CIP*, $SNMP_{OID\_LIST}$, $SNMP_{OID\_VALUE}$ and τ to *SNMP_TABLE*;
$ifInOctets_{new} = VAL1$;
$ifOutOctets_{new} = VAL2$;
$ifSpeed = VAL3$;
$\Delta_{ifInOctets} = ifInOctets_{new} - ifInOctets_{old}$;
$\Delta_{ifOutOctets} = ifOutOctets_{new} - ifOutOctets_{old}$;
$Bandwidth\_Utilization = \frac{max(\Delta_{ifInOctets}\ ,\ \Delta_{ifOutOctets}) * 8 * 100}{\Delta_{seconds} * ifSpeed}$;
if $(Bandwidth\_Utilization \geq Threshold_{Bandwidth})$ **then**
  CONGESTION_STATE = True;
  Intimate *CON*,*CIP*,*CTYPE*,*CCODE*;
else
  CONGESTION_STATE = False;
  Intimate *NOCON*,*CIP*,*CTYPE*,*CCODE*;
end if
Exit;

Algorithm 3.2 processes all the Net and Host Unreachable messages in the network. Net Unreachable messages are supposed to be received from the gateway. Hence, for any Net Unreachable message *NHU*, the first step is to check whether its source IP NHU_{IPS} matches with $IP_{GATEWAY}$. If there is a mismatch then spoofing is detected and an entry is made into *LOGT* table. Otherwise, if the *NHU* packet is destined for the *IDS*, it means that it is a reply to the probe that is sent by the IDS. In that case, event *NHUPRSP* is intimated to the detector module and the *CONGCHECK*() module is called. For the other Net/Host Unreachable messages, the IDS sends an ICMP Echo Request Probe *IPRQP* to NHU_{IPS} and invokes the *EXPHAND*() module with NHU_{IPS}, NHU_{type}, NHU_{code} passed as parameter. The flow chart of Algorithm 3.2 is given in Figure 3.2.

Algorithm 3.2 *NHUHAND*()

Input : *NHU* - ICMP Net/Host Unreachable packet
Output: Intimate receipt of *NHU*, *NHUPRSP* and sending of ICMP Echo probe request *IPRQP*

```
if NHU_IPS not in IP_GATEWAY then
    status ← spoofed;
    add {NHU_type, NHU_code, NHU_IPS, NHU_IPD, timeofreceipt} in LOGT;
else if NHU_IPD == IP(IDS) then
    Intimate receipt of NHUPRSP;
    Wait for t_icmp time;
    call CONGCHECK(IP_GATEWAY, NHU_type, NHU_code);
else
    Intimate receipt of NHU;
    Send ICMP Echo Probe Request IPRQP to NHU_IPS from IDS;
    Intimate sending of the ICMP echo probe request IPRQP;
    EXPHAND(NHU_IPS, NHU_type, NHU_code);
end if
Exit;
```

Algorithm 3.3 describes the *ECHOREPLYHAND*(). It takes as input all the ICMP Echo reply messages of the network but it processes only those packets that are destined for the IDS. Other echo reply messages are discarded as they do not help in detecting any of the ICMP attack. The flowchart for Algorithm 3.3 is given in Figure 3.3. In case of *NHU* attack detection, there will be no matching entry in *REDIRECTT*, thus *IPRSP* will be intimated.

Algorithm 3.4 describes *EXPHAND*() algorithm. The flowchart for Algorithm 3.4 is given in Figure 3.4. It takes as input, a host's *IP* (*EIP*), the type (*ETYPE*) and code (*ECODE*) of the *ICMP* packet for which it is called. A clock is triggered once the handler is called.

Algorithm 3.3 *ECHOREPLYHAND*()

Input : *IPRSP* - ICMP Echo Reply packet
Output: Intimate receipt of *IPRSP*, *IPRSPD*, *IPRSPP*

if $IPRSP_{IPD} == IP(IDS)$ **then**
  if $IPRSP_{IPS} == REDIRECTT_{IPS}[i]$ AND $IPRSP_{IPS} == REDIRECTT_{prefgw}[i]$ AND time of receipt of *IPRSP* - $REDIRECTT_{timeofsending}[i] < t_{icmp}$, for some i, $1 \leq i \leq REDIRECTT_{MAX}$ **then**
    $REDIRECTT_{count}[i] = REDIRECTT_{count}[i] + 1$;
    if $REDIRECTT_{count}[i] > 1$ **then**
      $status \leftarrow spoofed$;
      add $\{5, REDIRECTT_{code}[i], IPRSP_{IPS}, REDIRECTT_{IPD}[i], timeofreceipt\}$ in *LOGT*;
    else if $REDIRECTT_{TTL}[i] > IPRSP_{TTL}$ **then**
      Intimate receipt of *IPRSPP*;
      Wait for t_{icmp} time;
      Intimate $TTLdef$;
    else
      Intimate receipt of *IPRSPP*;
      Wait for t_{icmp} time;
      Intimate $TTLpref$;
      call $CONGCHECK(IP_{GATEWAY}, 5, REDIRECTT_{code}[i])$;
    end if
  else if $IPRSP_{IPS} == REDIRECTT_{IPS}[i]$ AND time of receipt of *IPRSP* - $REDIRECTT_{timeofsending}[i] < t_{icmp}$, for some i, $1 \leq i \leq REDIRECTT_{MAX}$ **then**
    $REDIRECTT_{count}[i] = REDIRECTT_{count}[i] + 1$;
    if $REDIRECTT_{count}[i] > 1$ **then**
      $status \leftarrow spoofed$;
      add $\{5, REDIRECTT_{code}[i], IPRSP_{IPS}, REDIRECTT_{IPD}[i], timeofreceipt\}$ in *LOGT*;
    else
      Intimate receipt of *IPRSPD*;
      Wait for t_{icmp} time;
      Send ICMP Probe Request *IPRQPP* to $REDIRECTT_{prefgw}[i]$ from IDS;
      add $\{REDIRECTT_{prefgw}[i], REDIRECTT_{code}[i], REDIRECTT_{prefgw}[i], REDIRECTT_{IPD}[i], timeofsending, 0, IPRSP_{TTL}\}$ in *REDIRECTT*;
      Intimate sending of the ICMP probe request *IPRQPP*;
      Call expiry handler $EXPHAND(REDIRECTT_{prefgw}[i], 5, REDIRECTT_{code}[i])$;
    end if
  else
    Intimate receipt of *IPRSP*;
  end if
end if
Exit;

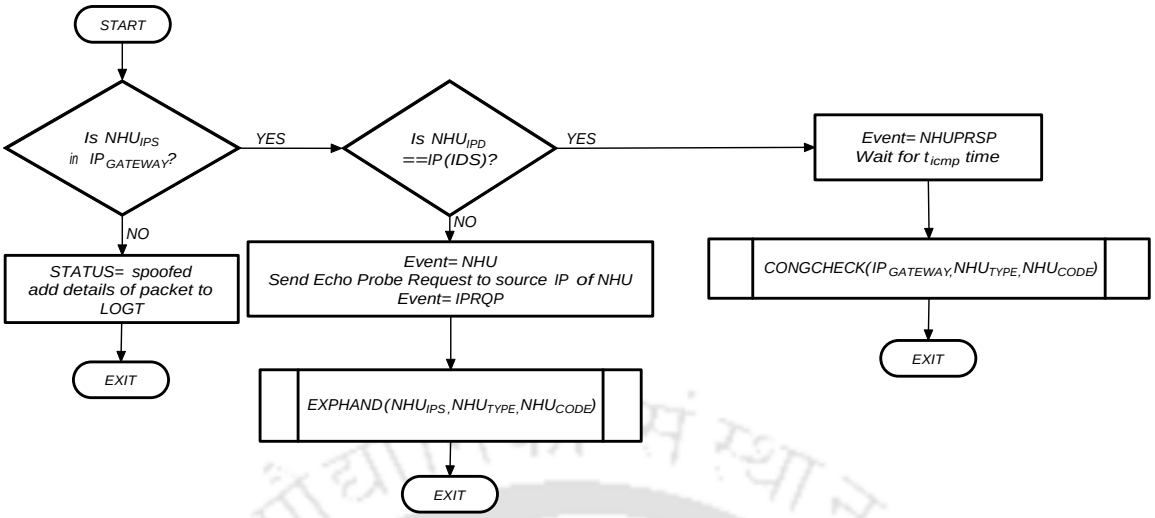

Figure 3.2: Flowchart for Net/Host Unreachable Handler

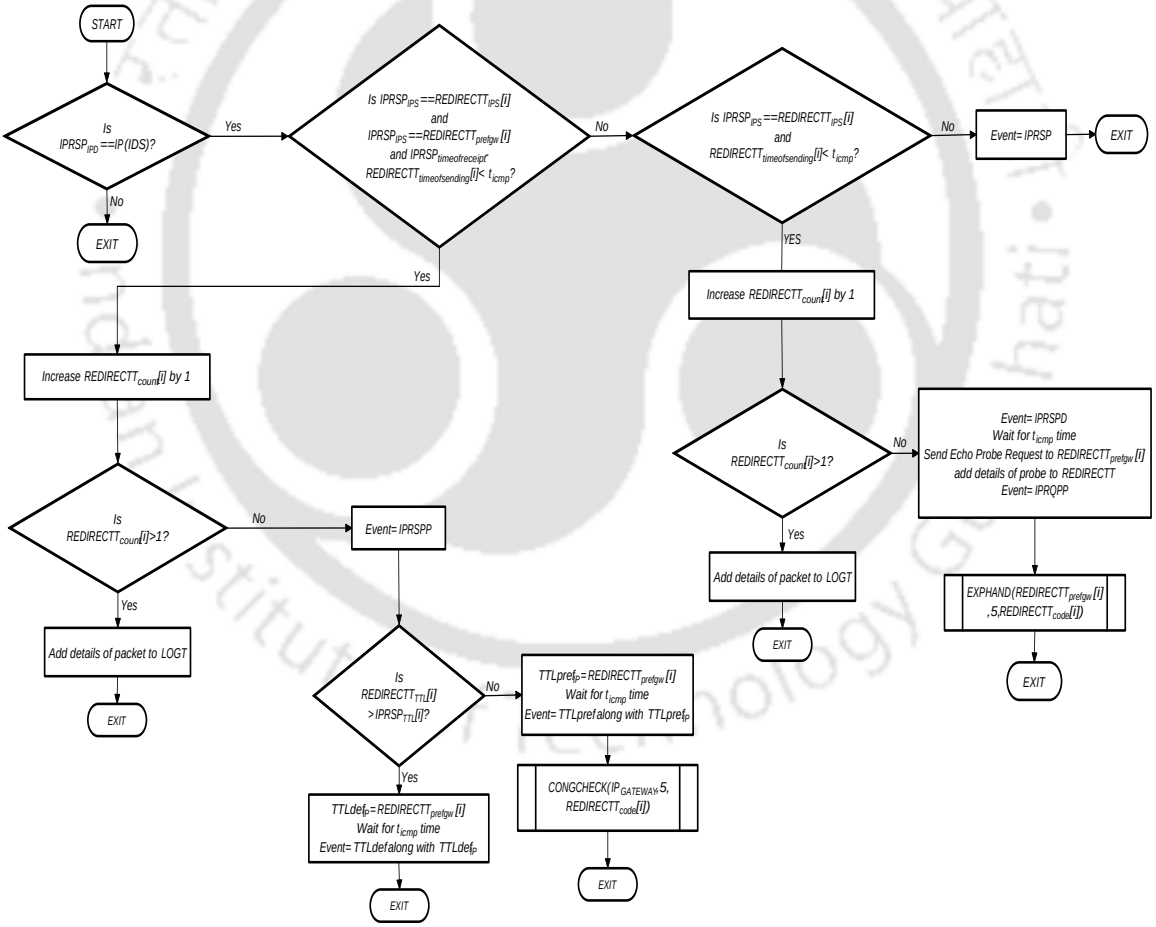

Figure 3.3: Flowchart for Echo Reply Handler

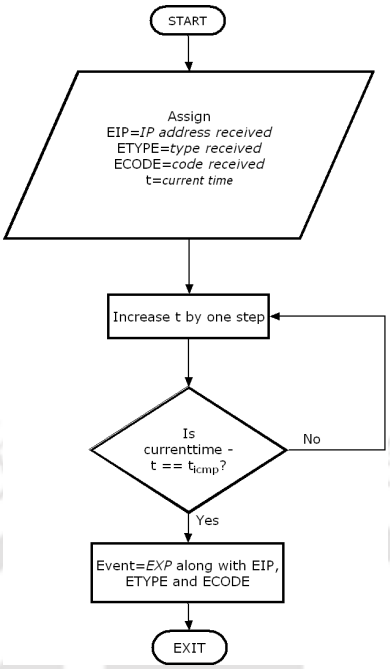

Figure 3.4: Flowchart for EXPHAND

Once the clock ticks for t_{icmp} amount of time, event *EXP* is intimated and the IP address, type and code that is received as parameter is returned.

Algorithm 3.4 *EXPHAND*(EIP,ETYPE,ECODE)

Input : IP address of a host, *type* of ICMP packet, *code* of ICMP packet.
Output: Intimate *EXP* and return *EIP*.

```
t ← currenttime;
while true do
    clock ticks up by one step;
    if currenttime − t == t_icmp (value of t_icmp is predefined) then
        Intimate EXP, EIP, ETYPE and ECODE;
    end if
end while
```

3.3.3 An Example

In this subsection an example is used to illustrate handling of spoofed *NHU* packets by the above handlers. Further, this example also highlights the difference in ICMP packet sequences (after active probing) in case of spoofing versus normal scenario. The network architecture used in this example has already been explained earlier. Figure 3.5 shows the sequence of packets (indicated with packet sequence numbers).

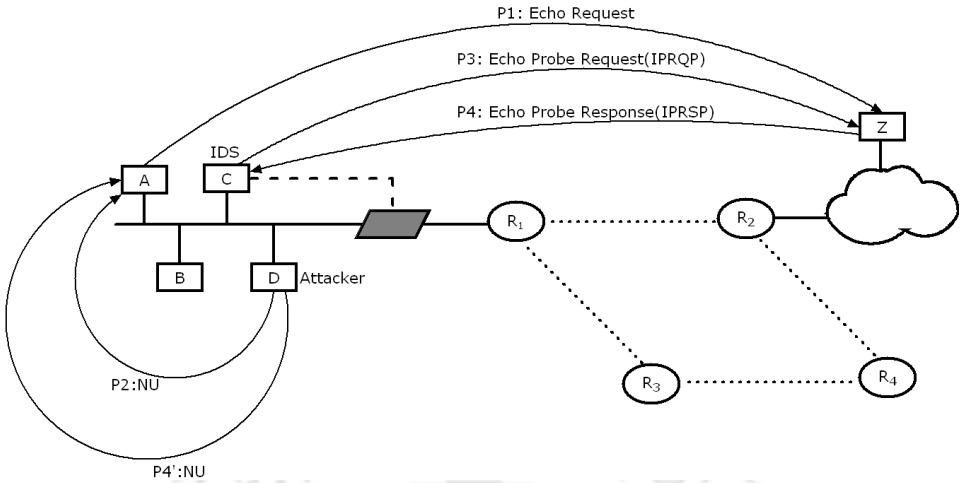

Figure 3.5: Example for Net/Host Unreachable

- Packet Sequence (P1): Suppose A wants to communicate with host Z in another subnet. A pings host Z and sends an **ICMP echo request** to it.

- Packet Sequence (P2): Let us suppose that host D is the attacker who wants to prevent A from communicating with Z. Thus D responds to this Echo request by sending a spoofed net/host unreachable (*NHU*) packet to the host A.

- Packet Sequence (P3): The IDS receives a copy of this *NHU* packet (P2) and *NHUHAND*() is invoked as a result. In the genuine case, router *R*1 would have sent the *NHU* packet. However in attack case, D spoofs the packet as if it is coming from *R*1. Thus, NHU_{IPS} is present in $IP_{GATEWAY}$. Destination of the packet is A, not the IDS. Therefore the first two conditions of the handler are not matched. So the handler sends **ICMP Echo Probe Request** to D and intimates sending of probe *IPRQP*. Also, a call to *EXPHAND*() is made to ensure that only probe responses that arrive within t_{icmp} time are dealt with. This probing creates a difference in packet sequences between normal and attack conditions.

- Packet Sequence (P4): Within t_{icmp} time Z will respond to this ICMP Probe request (P3) with an **ICMP Echo Reply**. This event would help the detector to identify the attack.

- Packet Sequence (P4'): Now there is a possibility that the reply does not arrive within t_{icmp} period. Instead, another Net unreachable is received. *NHU* handler calls the

CONGCHECK() to get the congestion status. If congestion does not exist, it means that Z's network is truly unreachable, otherwise no conclusion is reached. The detector module makes the decision and alerts the system administrator in case of attack.

As discussed before, the IDS engine for detecting the ICMP attacks would be constructed using a DES detector. So initially the ICMP events under normal and spoofed conditions are modeled using state-event based DES models and subsequently a DES detector is designed.

3.3.4 DES Modeling

The DES model used for representing the system under normal and attack scenarios is a five tuple $\langle X, X_0, \Sigma, V, \delta \rangle$, where:

- X is the set of states.
- $X_0 \subseteq X$ is the set of initial states.
- Σ is the set of events.
- V is the set of model variables. Each element v_i of V ($V = \{v_1, v_2, \cdots, v_n\}$) can take values from a domain D_i.
- $\Im$ is the set of transitions. A transition $\tau \in \delta$ is a five tuple $\langle x, x', \sigma, check(V), assign(V) \rangle$, where x is the source state, x' is the destination state, σ is an event (on which the transition is fired), $check(V)$ is boolean conjunction of equalities of a subset of variables in V and $assign(V)$ is a subset of model variables and assignments with values from their corresponding domains.

Representation $G_N : \langle X_N, X_{0N}, \Sigma_N, \Im_N, V_N \rangle$ is used for a DES model under normal conditions and $G_{Fi} : \langle X_{Fi}, X_{0Fi}, \Sigma_{Fi}, \Im_{Fi}, V_{Fi} \rangle$ is used for DES model under failure(attack) Fi condition. Here i is used to denote i-th failure type, as there can be multiple failures and more than one DES attack models corresponding to those. In this chapter for every system model there is a single type of failure model denoted by G_{f1} but multiple failures are considered while describing the proposed algorithms.

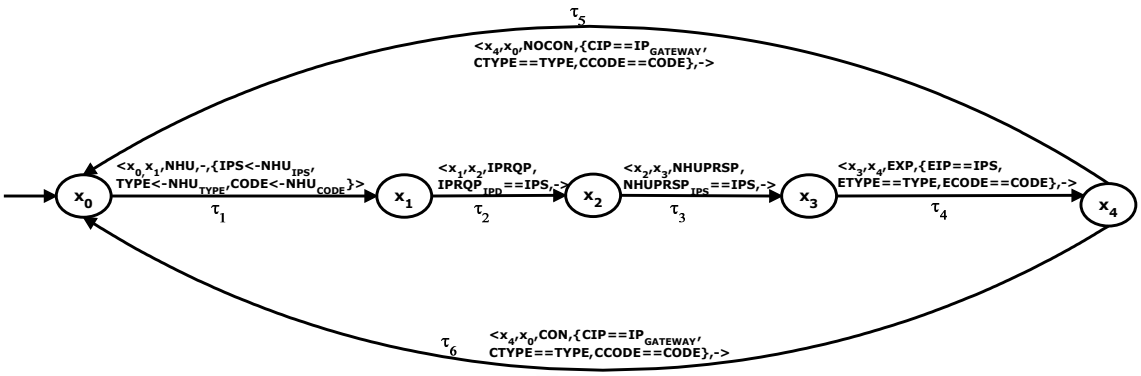

Figure 3.6: DES Model under normal condition

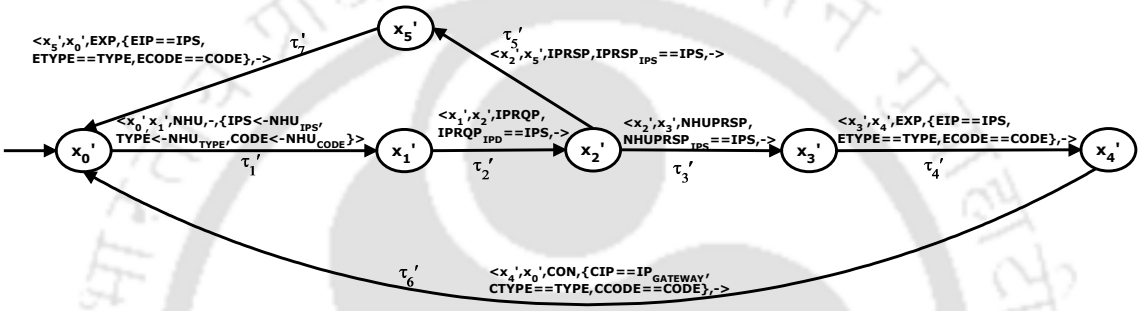

Figure 3.7: DES Model under attack condition

Now, DES modeling of the network during NHU attack is exhibited. Figure 3.6 shows the normal DES model G_N of the network. Figure 3.7 shows the DES model G_{F1} under NHU attack scenario. It may be noted that there is no final state in the DES model as ICMP traffic in network is a continuous (or renewal) process (never halts as long as network is up). There are some renewal states from which there is a transition to an initial state(s). Thus, the system is *live*.

The terms required for DES modeling are quantified as follows:
$\Sigma = \{NHU, IPRQP, NHUPRSP, IPRSP, EXP, CON, NOCON\}$. The set of states X is shown in Figure 3.6 and Figure 3.7. States with no primes correspond to normal situation and those with single primes denote attack condition respectively. Initial states (x_0,x_1) are also shown in Figure 3.6 and Figure 3.7. Model variable set is $V = \{IPS, TYPE, CODE\}$; IPS has the domain as $D_1 = \{y.y.y.y | 1 \leq y \leq 255\}$; $TYPE$ has the domain as $D_2 = \{y | 0 \leq y \leq 7\}$; $CODE$ has the domain as $D_3 = \{y | 8 \leq y \leq 15\}$. The transitions are also shown in Figure 3.6 and Figure 3.7, transitions without primes are for normal model while single primes are for attack. It may be noted that there are "−" for some fields in the tuple representing the

transitions. If " − " is for *check*(V) then it represents TRUE condition, while if the " − " is for *assign*(V) then it represents NO action (i.e., assignment) is required. An overview of the DES model for the NHU attack scenario (Figure 3.7) is explained as follows.

The model moves to state x'_1 on observation of an event *NHU* by transition τ'_1. Enabling of τ'_1 is only dependent on *NHU* and not on any model variables; this is depicted by " − " in the transition. Model variables *IPS*, *TYPE* and *CODE* are assigned with source IP, type and code of the ICMP *NHU* packet respectively. Following that in state x'_2, ICMP probe is sent. Transition τ'_2 is enabled on *IPRQP* (sent to source IP of the *NHU* under consideration); correspondence of *IPRQP* with *NHU* is determined by checking if model variable *IPS* matches with $IPRQP_{IPD}$. After transition τ'_2, there are two options; (i) either probe response from genuine host arrives (τ'_5) in the form of Echo Reply or (ii) probe response (τ'_3) arrives in the form of another *NHU* packet. Now, in the first case, it becomes obvious that the initial *NHU* packet was spoofed. τ'_7 is triggered by the *EXP* event indicating that no more waiting for a response is required and the system renews. In the second case i.e. when *NHUPRSP*s is received as probe response, the initial *NHU* packet is mostly genuine. But there is a possibility that there is congestion in the network because of which the genuine host is unable to respond, and meanwhile the attacker has forged the response *NHUPRSP* (Net/Host unreachable message with spoofed IP). In order to resolve this issue, the *CONGCHECK*() is used. If there is no congestion *NOCON*, it is concluded that the message is surely genuine. However, in case of *CON* event, the decision becomes ambiguous. So transition τ'_6 is added in the attack model. τ'_6 takes the control to the initial state, similar to τ'_7. These are renewal transitions.

3.3.4.1 Detector

After development of the normal and attack DES models, a detector is designed which can identify whether a given sequence of events correspond to normal or attack scenario. Detector is basically a kind of state estimator that estimates which state to go when an event is given to it. Finally, the detector declares an attack when a state (of the detector) is reached whose estimate comprises states only from the attack model, after following a complete event trace. Before the detector is formally discussed the following definitions are introduced:

Definition 3.1 *Two transitions* $\tau_1 = \langle x_1, x'_1, \sigma_1, check_1(V), assign_1(V)\rangle$ *and* $\tau_2 = \langle x_2, x'_2, \sigma_2, check_2(V), assign_2(V)\rangle$ *are equivalent if* $\sigma_1 = \sigma_2$ *(same event),* $check_1(V) \equiv check_2(V)$ *(same equalities over the same subset of variables in V), and* $assign_1(V) \equiv assign_2(V)$ *(same subset of model variables with same assignment).*
If $\tau_1 \equiv \tau_2$ *then the source states of the transitions are equivalent and so are the destination states, i.e.,* $x_1 \equiv x_2$ *and* $x'_1 \equiv x'_2$.

The detector is represented as a directed graph $O =< Q, A >$ where,

- Q is the set of detector states, called *O-states*.
- A is the set of detector transitions, called *O-transitions*.

Henceforth, terms like transitions, states etc. of the DES are termed as model transitions, model states etc. to differentiate them from those of the detector. Each O-state $q \in Q$ comprises a subset of equivalent model states representing the uncertainty about the actual state and each O-transition $a \in A$ is a set of equivalent model transitions representing the uncertainty about the actual transition that occurs. The initial O-state comprises all initial model states. Following that, given any O-state q, the O-transitions emanating from q are obtained as follows. Let $\mathfrak{I}_q$ denote the set of model transitions from the model states $x \in q$. Let A_q be the set of all equivalence classes of $\mathfrak{I}_q$. For each $a \in A_q$, an O-transition is created comprising all model transitions in a. Then the successor O-state for a is constructed as $q^+ = \{x | x$ is the destination model state of $\tau \in a\}$. Successor for O-states determining attack or normal scenarios are not created. This process is repeated till no new O-transition can be created.

Definition 3.2 *An O-state, which contains only model states corresponding to normal situation is called Normal certain O-state.*

Definition 3.3 *An O-state, which contains only model states corresponding to attacks is called Attack certain O-state.*

Definition 3.4 *An O-state which contains model states corresponding to both normal and attack situation, is called an Uncertain state.*

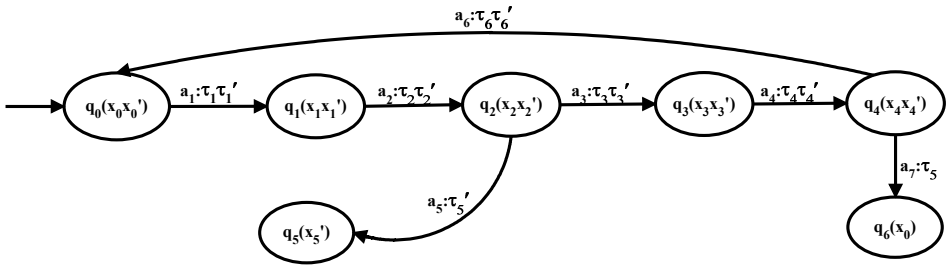

Figure 3.8: Detector for DES Model of Figure 3.6 and Figure 3.7

Normal/attack certain O-states denote that the current (model) state estimate comprises only normal/attack states, thereby making a decision. On the other hand, an uncertain in O-state does not confirm any decision. It implies that there is path both in the normal and attack model which produce the same record of transitions.

Definition 3.5 *An attack-indeterminate cycle in O is a cycle composed of only uncertain O-states.*

If an attack-indeterminate cycle exists in the detector then no decision can be made within a finite delay after the attack has occurred as the detector may loop in that cycle forever.

Definition 3.6 *A system is diagnosable with respect to an attack, if there are no attack-indeterminate cycles in the detector. It is undiagnosable otherwise.*

The algorithm to construct the detector is presented in Algorithm 3.5.

Figure 3.8 illustrates the detector for the DES models in Figure 3.6 and Figure 3.7. Some of the initial steps for this detector are as follows.

- The initial state of the detector i.e., q_0 comprises all initial model states x_0, x_0'.
- $\Im_{q0} = \{\tau_1, \tau_1'\}$ which are all the outgoing model transitions from model states in $q_0 = \{x_0, x_0'\}$. Now, $A_{q_0} = \{\tau_1, \tau_1'\}\}$, as $\Im_{q_0}$ can be partitioned into one equivalent set. Corresponding to $\{\tau_1, \tau_1'\}$ there is an O-transition a_1 emanating from q_0.
- The destination O-state corresponding to a_1 is $q_1 = \{x_1, x_1'\}$ as the destination model state for τ_1 and τ_1' is x_1 and x_1', respectively.

Algorithm 3.5 Construction of detector O for the DES model

Input: DES models G_N for Normal and G_{Fi} where $1 \leq i \leq f$ for Attack conditions. /* Number of failure types is assumed to be f. */

Output: Detector for IDS

Begin
$q_0 \leftarrow X_{0N}$; /* initial states of normal model */
FOR ALL i, $1 \leq i \leq f$ /* number of failure types */
  $q_0 \leftarrow q_0 \cup X_{0F_i}$;
$Q \leftarrow q_0$;
$A \leftarrow \phi$;
FOR ALL $q \in Q$ which are not normal or attack certain **Do** {
  $\Im_q \leftarrow \{\tau | \tau \in \Im \wedge$ source state of $\tau \in q\}$;
  Find the set of all equivalent classes A_q, of $\Im_q$;
  FOR ALL $a \in A_q$ {
    $q^+ = \{x | x$ is destination state of $\tau \in a\}$;
    $Q = Q \cup \{q^+\}$;
    IF q^+ is attack certain , {
      $E \leftarrow \sigma$, σ is the event for model transition $\tau \in a$
      where a is the O-transition emanating from q and in the path leading to q^+;
      Add E_{type}, E_{code}, E_{IPS}, E_{IPD} and Time of receipt of E to $LOGT$;
    } /* Entry of tables for certain states */
    $A = A \cup \{a\}$;
  } /* For all $a \in A_q$ */
} /* for $q \in Q$ which are not certain */
End

In the detector, states q_5 is attack certain O-state and q_6 is normal certain state.

The issue in the detector is presence of the attack-indeterminate cycle q_0, q_1, q_2, q_3, q_4 in it. This leads to undiagnosability as per the definition of diagnosability. Presence of this cycle of uncertain states means that attack may have occurred somewhere in between the states q_0 to q_4 but the detector can loop in cycle forever without ascertaining occurrence of the attack. Attack can be confirmed only after transition a_5. So the overall fault diagnosis is not possible, but it can be done in some traces. To detect the occurrence of the fault an alternative partial diagnosis framework is needed using which the traces where attack certain O-states are visited (e.g. q_6) can be diagnosed.

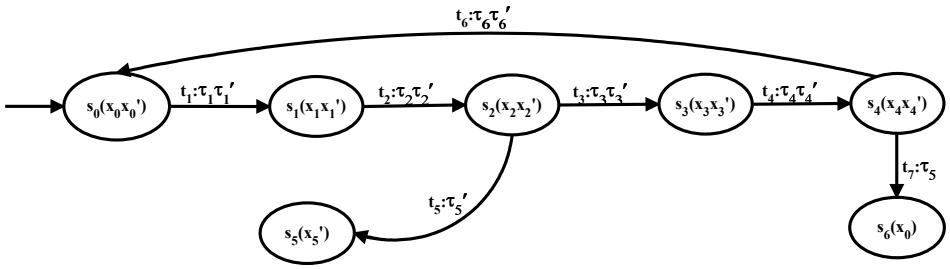

Figure 3.9: I-Detector for DES Model of Figure 3.6 and Figure 3.7

3.3.5 I-detector

In this section the concept of partial diagnosis is briefly explained and the partial detector (*I*-detector) is built for NHU attack. Before describing the procedure in detail, concept of partial diagnosis is explained as stated in [20]. The preceding definition of diagnosability requires the absence of attack-indeterminate cycle in all traces generated by detector *O*. Relaxed definition of diagnosability termed as I-diagnosability, requires the diagnosability condition to be valid only in those traces where an attack-uncertain *O*-state is followed by an indicator transition, not in every trace containing failure. Thus a failure is associated with one or more indicator transitions. The events triggering the indicator transitions are indicator events. If a fault has occurred prior to an indicator event, then that fault must be detected within a finite delay after its occurrence. Therefore a system is I-diagnosable if after an indicator event, there is no cycle of attack-uncertain *O*-states in any trace. This relaxed definition helps to perform partial diagnosis of a system in some paths where indicator events are defined, even if the system as a whole is found to be undiagnosable.

The *I*-detector is constructed by appending a labeling function to the directed graph of detector *O*. Thus $I =< S, T, L >$ where,

- S is the set of *I*-detector states, called *I-states*. This set is same as that of detector states Q.
- T is the set of *I*-detector transitions, called *I-transitions*. This set is also same as that of detector transitions A.
- L is labeling function used to label the *I*-states. $L(s) = f - i$, if s is an attack-uncertain state and the trace from the start state leading to s contains the corresponding indicator event. $L(s) = nothing$, otherwise.

Definition 7 A I-state s is called an f-i state if label of state s is $L(s) = f$-i.
Definition 5 An f-i indeterminate cycle in I is a cycle composed of only f-i states. This means that there is a cycle both in the normal model and attack model which produce same record of transitions and the cycle in the attack model has an indicator event.

Definition 8 A system is I-diagnosable if there is no f-i indeterminate cycle in I. Presence of such a cycle implies that it will not be possible to detect the occurrence of the fault even in those traces that contain the indicator, as the system may continue to loop in those states forever.

Now in case of the detector of NHU attack this partial diagnosis can be done. The indicator event is chosen as *NOCON* as it is quite a natural fact that no fault diagnosis is possible in a congested network. Event *NOCON* implies that *CONCHECK*() has confirmed that there is no congestion in the network. Using this indicator event the I-detector is built as shown in Figure 3.9. The construction procedure is similar to that of detector, apart from proper labeling as defined earlier. To avoid repetition the detailed description of that is omitted.

It can be noticed that there is no f-i indeterminate cycle in the I-detector of Figure 3.9, though there is an attack indeterminate cycle $s_0, s_1, s_2, s_3, s_4, s_0$. In that attack indeterminate cycle there is no occurrence of a transition triggered by indicator event *NOCON*. So it does not violate the condition of I-diagnosability. The fault certain trace, s_0, s_1, s_2, s_5 is normally diagnosable. The trace where indicator transition τ_5 occurs is a normal certain trace $s_0, s_1, s_2, s_3, s_4, s_6$. The diagnosis process can continue in these two traces as per the definition, leaving out the indeterminate cycle. Now detection of attack scenario and normal scenario for NHU packet is discussed.

Attack Scenario

Let detector reach state s_5 by the sequence s_0, s_1, s_2. In the states s_1, s_2, s_3 attack or normal condition cannot be determined as the state estimate comprises one normal model state and one attack model state, e.g, s_1 has x_1 and x'_1. As s_5 is an attack certain I-state, reached by occurrence of t_5 (by virtue of τ'_5), I-detector declares an attack. It may be observed

from Figure 3.7 that τ'_5 corresponds to a *IPRSP* for a *IPRQP* (τ'_2) within t_{icmp} time. t_1 is the *I*-transition emanating from s_0 and is in the path leading to s_5. The event corresponding to t_1 is (same as τ_1/τ'_1) is *NHU* indicating receipt of a spoofed *NHU* packet. So, in s_5 details of this attack packet are logged in *LOGT* table.

Normal Scenario

Let detector reach state s_6 by the sequence s_0, s_1, s_2, s_3, s_4 (involving no normal/attack certain states). s_6 is a normal certain *I*-state, reached by occurrence of t_7 (by virtue of τ_5), thus *I*-detector declares the packet being verified as normal. It may be observed from Figure 3.6 that τ_5 corresponds to the fact that *NHUPRSP* has arrived for the *IPRQP* within t_{icmp} time and following that no congestion has been detected in the network.

Looping in cycle of uncertain states scenario

I-detector may loop in sequence $s_0, s_1, s_2, s_3, s_4, s_0$ (involving no normal/attack certain *I*-states). In this case attack cannot be detected within a finite delay even if it has already occurred. But there is no indicator transition (*NOCON*) occurring in this cycle. So the diagnosability condition can be relaxed for this trace without hurting the overall notion of I-diagnosability.

3.4 Partial diagnosis Using Reduced I-detector

It can be seen from the partial diagnosis done in the previous section that the *I*-detector constructed for NHU attack contains an unnecessary path (i.e. $s_0, s_1, s_2, s_3, s_4, s_0$), where diagnosis is not possible. In a large complex system there might be many such traces that are undiagnosable but the *I*-detector will contain those. This provides the motivation to create a reduced *I*-detector (called *RI*-detector) that would contain only those paths where fault diagnosis can be performed. In this section the algorithm for constructing this reduced detector is proposed. The running time of this detector is proven to be less than *I*-detector and the working of *RI*-detector is shown for ICMP attacks.

3.4.1 Reduced I-detector

In the detector of Figure 3.9, s_0, s_1, s_2, s_5 is a trace where attack can be confirmed. The detector moves for the first time from an uncertain state to an attack-certain state following the transition a_5. Such a transition is called an AD-transition (Attack Detecting Transition). Transition a_5 corresponds to τ'_5 from the attack model but it does not have any corresponding transition from the normal model. Thus using these AD-transitions fault is always diagnosable. A machine can be designed to detect occurrence of such AD transitions. This machine is called *RI-detector*.

In a typical complex system number of states and transitions can be very high. It is worthwhile to design the *RI*-detector that can detect whether there is an attack by only monitoring the AD-transitions instead of the entire set of transitions. The detection of an AD-transition is a two step procedure. In the first step the *RI*-detector checks whether the received transition takes the system to a state from which a AD transition can emanate. If it does so, in the next step the *RI*-detector checks whether the following transition is equivalent to the AD-transition emanating from that state. If it so happens then the detector concludes that a fault has occurred. Note that the set of AD-transitions is synthesized beforehand. Also, both the steps can be performed only by looking at the system and the set of AD-transitions. A detailed algorithm for constructing the *RI*-detector and forming the set of AD-transition is explained in the next subsection.

3.4.2 Construction of RI-detector

Before describing the construction procedure of *RI*-detector, the AD-transitions and *RI*-detector are formally characterized.

Definition 8 A transition a of detector is a AD-transition if, the source state of a is an uncertain O-state and the destination state of a is an attack certain O-state.

An *RI*-detector is formally presented as a directed graph

$$RI = < M, B, Ou > \text{where,}$$

- M is the set of states called *RI*-state.
- B is the set of transitions called *RI*-transitions.

- Ou is the output function defined as $Ou(b) \rightarrow \{Y, N\}$, where b is an RI-transition.

Each RI-transition b is a three tuple $\langle s, t, o_{val} \rangle$ where s is an O-state, t is an O-transition and o_{val} is the output value generated by Ou; output Y conforms a fault whereas N denotes that fault can not be confirmed.

In general there are three types of states in an RI-detector - an initial state m_0, an intermediary state for every AD-transition, $m_1, m_2, ..., m_l$ (assuming $\Im_{AD} = \{a_1, a_2, ..., a_l\}$ is set of AD-transitions) and a single final state m_f. Let τ'_i be the corresponding transition in the attack model for an AD-transition $a_i \in \Im_{AD}$. Whenever the attack model moves to a state x' such that x' is the source state of a transition τ'_i, the RI-detector moves to the intermediary state m_i. Thus, there is a transition from m_0 to m_i given by $\langle source\ state\ of\ a_i, -, N \rangle$. "$-$" denotes do not care signifying that no check is required on the transition taken by the system. For other transitions it loops on the initial state with $o_{val} = N$. Now at m_i, if the attack model moves to a state x' such that x' is the destination state of transition τ'_i, the RI-detector moves to final state m_f. The transition from m_i to m_f is given as $\langle destination\ state\ of\ a_i, t_i, Y \rangle$. If system executes any other transition, the RI-detector moves back to initial state m_0 with $o_{val} = N$. Once the detector reaches m_f state it stays there forever representing a permanent failure.

As stated earlier, to construct the RI-detector, first the set of AD-transition is needed. This set of AD-transition can be obtained from the detector, but construction of the entire detector is not required. A closer inspection reveals that only the uncertain and attack-certain states are needed, leaving out the normal certain state. The normal and attack models are the inputs. The set of AD-transitions is obtained from them using Algorithm 3.6.

In case of NHU attack it can be seen from Figure 3.8 that the initial state q_0 comprising $\{x_0, x'_0\}$ is an uncertain state. It is added to the set of states S. From q_0 following event NHU it moves to another uncertain state q_1 comprising $\{x_1, x'_1\}$. Similarly it moves to state q_2 after that. All these uncertain states are added in S. From q_2 following transition a_5 emanates a path that leads to attack certain state q_5 comprising x'_5. So transition a_5 is added to the set of AD-transitions τ_{AD}. There are no other such transition having an uncertain source state and attack-certain destination state. Thus when the algorithm terminates, τ_{AD} contains one single transition a_5 having source state q_2 and destination state q_5. Once this set τ_{AD} is obtained, RI-detector can be constructed using Algorithm 3.7.

Algorithm 3.6 Finding the set of AD transitions

Input: DES models G_N for Normal and G_{Fi} where $1 \leq i \leq f$ for Attack conditions
Output: The set of AD transitions $\Im_{AD}$

Begin
$s_0 \leftarrow X_{0N}$; /* initial states of normal model */
FOR ALL i, $1 \leq i \leq f$ /* number of failure types */
  $s_0 \leftarrow s_0 \cup X_{0F_i}$; /* initial states of each failure models */
$S \leftarrow s_0$;
$T \leftarrow \phi$;
FOR ALL $s \in S$ which are uncertain **Do** {
  $\Im_s \leftarrow \{\tau | \tau \in \{\Im_N \cup \Im_{Fi} \wedge$ source state of $\tau \in s\}$;
  Find the set of all equivalent classes T_s, of $\Im_s$;
  FOR ALL $t \in T_s$ {
    $s^+ = \{x | x$ is destination state of $\tau \in t\}$;
    IF s^+ is an uncertain state **THEN**
      $S = S \cup \{s^+\}$;
    IF s^+ is attack certain **THEN**
      $\tau_{AD} \leftarrow \tau_{AD} \cup t$;
    $T = T \cup \{t\}$;
  } /* For all $t \in T_s$ */
} /* for $s \in S$ which are not certain */
End

The states m_0 and m_f will always be contained by an *RI*-detector. So these two states are added first. Then for every AD-transition present in $\Im_{AD}$, first a state m_i is created. Then proper transitions are added to and from m_0 and m_f to the state m_i as explained earlier. This is done for every AD-transition. Finally self-loop transitions are added on m_0 and m_f with proper outputs. This concludes the construction of the *RI*-detector.

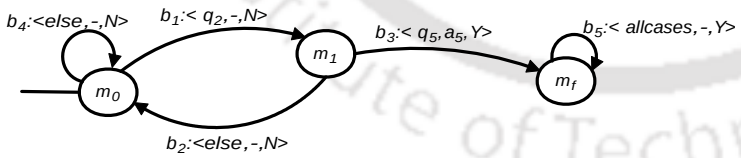

Figure 3.10: RI-detector for NHU attack

Attack detection scenario: The *RI*-detector for NHU attack is shown in Figure 3.10. This *RI*-detector is designed to detect the occurrence of AD-transition a_5. It starts from state m_0, it reaches state m_1 by transition b_1 whenever the system moves to state x'_2 i.e. the source state of τ'_5 contained in a_5. The value dont care in transition check signifies that this transition is taken as and when the next state is estimated to be x'_2, without looking at

Algorithm 3.7 Constructing the *RI*-detector

Input: The set of AD-transitions τ_{AD} built in previous algorithm.
Output: The *RI*-detector.

Begin
$M \leftarrow \{m_0, m_f\}$;
$B \leftarrow \phi$;
$i \leftarrow 1$;
FOR ALL $t \in \tau_{AD}$ **Do** {
  $M \leftarrow M \cup \{m_i\}$;
  $o_{val_i} \leftarrow Ou(b_i) = N$;
  $b_i \leftarrow \{source\ state\ of\ t, -, o_{val(i)}\}$;
  Add transition b_i between states m_0 and m_i;
  $o_{val_{i+1}} \leftarrow Ou(b_{i+1}) = N$;
  $b_{i+1} \leftarrow \{else, -, o_{val(i+1)}\}$;
  Add transition b_{i+1} between states m_i and m_0;
  $o_{val_{i+2}} \leftarrow Ou(b_{i+2}) = Y$;
  $b_{i+2} \leftarrow \{\ destination\ state\ of\ t, t, o_{val(i+2)}\}$;
  Add transition b_{i+2} between states m_i and m_f;
  $B \leftarrow B \cup b_i \cup b_{i+1} \cup b_{i+2}$;
  $i \leftarrow i + 3$;
}
$b_i \leftarrow \{else, -, \}$;
$O(b_i) \rightarrow N$;
$i \leftarrow i + 1$;
$b_i \leftarrow \{allcases, -\}$;
$O(b_i) \rightarrow Y$;
End

input transition. The output value of the transition is N as the detector cannot confirm the occurrence of fault at this point. If the next state had been some state other than x'_2, then a self-loop transition b_4 occurs. The transition b_3 from m_1 indicates that transition τ'_5 is taken by the system as its next transition. This causes detector to move from state m_1 to m_f outputting Y as the occurrence of fault can be confirmed after an AD-transition. If the input does not match with a AD-transition, the detector takes transition b_2 and goes back to initial state m_0. Once it reaches m_f, on every input it stays in that particular state forever and output is always Y. It signifies the fact that a fault is always permanent.

3.4.3 Complexity Analysis

It can be seen that if the set of AD-transition is given, then *RI*-detector can be constructed from that using Algorithm 3.7 by touching every transition of the set only once. It takes linear time to the number of AD-transitions. In worst case all the transitions in the system can be AD-transitions. In that case the running time of Algorithm 3.7 will be in order of the square of the system states. But finding the set of AD-transitions using Algorithm 3.6 has exponential worst case time complexity. Algorithm 3.6 requires to construct the detector in first place. Though the complete detector is not required as normal paths can be omitted as stated earlier, but in worst case if no normal trace is found then it will be equivalent of constructing the complete detector. In [20] authors have shown that the worst case complexity of constructing an detector is exponential to the number of system states. Thus in the overall construction process of *RI*-detector, Algorithm 3.6 dominates Algorithm 3.7 complexity-wise. So *RI*-detector construction is of order of exponential to the system states.

Comparing the *RI*-detector (Figure 3.10) and the *I*-detector (Figure 3.8), it may be noted that the area of the DES detector is much more than the *RI*-detector; while the former has 7 states and 8 transitions, the latter has only 3 states and 5 transitions. This fact can also be appreciated from the observation that the number of nodes in a DES detector can be exponential to the number of system states [20]. Instead in an *RI*-detector the number of states is in the order of the number of system transitions (when all system transitions are AD-transitions) which is only in order of the square of system states. So the running time of the *RI*-detector will be of order of square, whereas running time of *I*-detector will be of order of exponential to the number of system state. Thus although the construction of *I*-detector and *RI*-detector are of same order, once the *RI*-detector is built, it will run in much lesser time complexity than *I*-detector.

3.5 Experimentation and Results

The test bed created for our experiments is shown in Figure 3.1. It consists of four machines (A-D) in Subnet 1 and another machine in Subnet 2, running different operating systems. Machines A-D are running the following OSs: Windows XP, Ubuntu, Backtrack 4 and

Backtrack 4, respectively.The machine D with Backtrack 4 is acting as the attacker machine and machine C is set up as the IDS. These machines are connected in a LAN with a CISCO catalyst 3560 G series switch [57] with port mirroring enabled for system C. Subnet 2 contains the machine Z which is running Ubuntu OS. We use four Routers R1, R2, R3 and R4 to build our network setup. SNMP and NetFlow are enabled in Router R1 which acts as the gateway for Subnet 1. Linux machines with two or three network interface cards are used as routers. They have Quagga [58] routing software suite installed in them along with the Net-SNMP package [59] for the SNMP support. The zebra daemon of quagga handles the basic routing functionality. Also, IP forwarding is enabled in these machines to enable them act as routers. The tables are created in MySql database. The IDS is designed using C++ language. The IDS has two preemptive modules namely, packet grabber and packet injector. Packet grabber sniffs the packets from the network, filters ICMP packets and invokes the respective algorithm depending upon the packet type. Packet injector generates various probes necessary for verification of the ICMP messages. Attack generation tools Ettercap, Sing, Hping2 and other attack codes in C were deployed in machine D and several scenarios of spoofing ICMP messages were attempted.

Table 3.1 shows detection rate versus percentage of packets being probed for ICMP NHU based attack detection. At 100% probing, the detection rate is close to 98%. The rest 2% cases were not detected due to congestion in the network. At 20% of probing, the detection rate falls to around 37%. The accuracy is 100% in all cases and the detection rate is proportional to the tradeoff in probing.

Table 3.1: ICMP NHU based attack detection statistics

% of probe packet	no. of attacks launched	Detection rate
100	160951	97.87
80	160951	88.57
60	160951	71.32
40	160951	58.21
20	160951	37.15

Figure 3.11 and Figure 3.12 show the amount of extra ICMP traffic generated due to the probes sent for verifying ICMP messages. The plot in Figure 3.11 shows the amount of traffic under normal operation in the presence and absence of the IDS. It is noticed that the extra traffic generated due to probing is negligible. Plot in Figure 3.12 shows the traffic

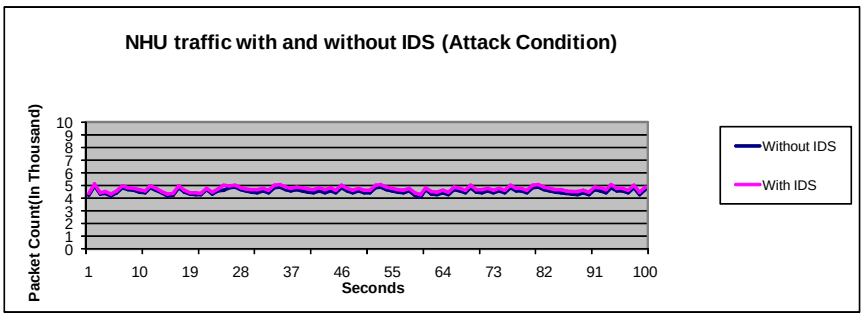

Figure 3.11: Traffic at ICMP NHU Normal Condition

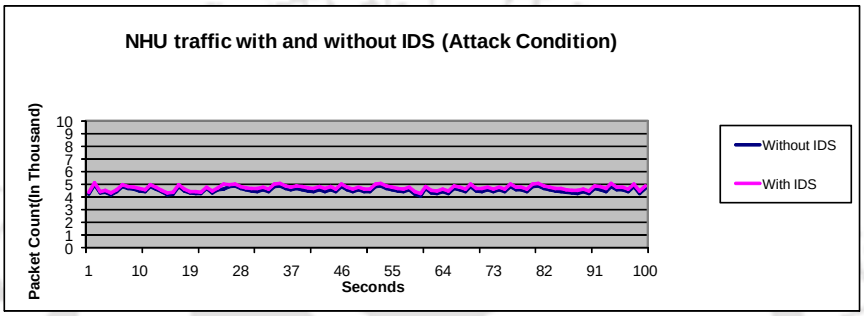

Figure 3.12: Traffic at ICMP based Attack Condition

generated when IDS is running and there are NHU message based attacks in the network. In this case, also small amount of extra traffic can be observed which is generated by the IDS due to probes. Considering the small percentage of ICMP traffic in network traffic, no significant traffic increase is observed.

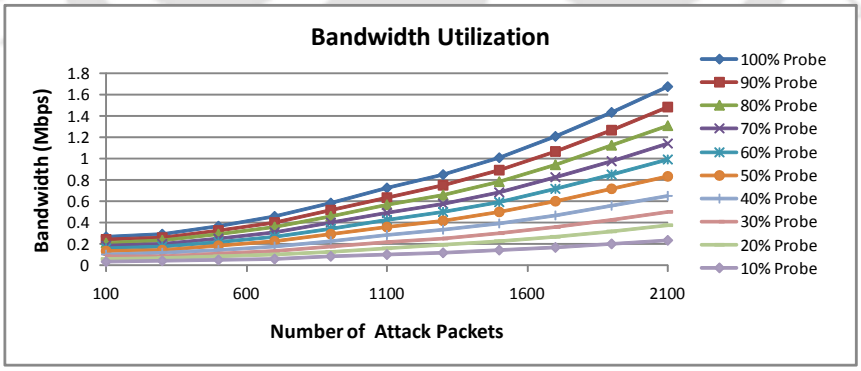

Figure 3.15: Bandwidth Utilization of IDS

In the testbed, we injected upto 10000 attack packets over a period of 100 seconds and measured CPU utilization of the processor running the IDS and memory utilization of the host executing the IDS and bandwidth utilization. Figure 3.13, Figure 3.14 and Figure 3.15

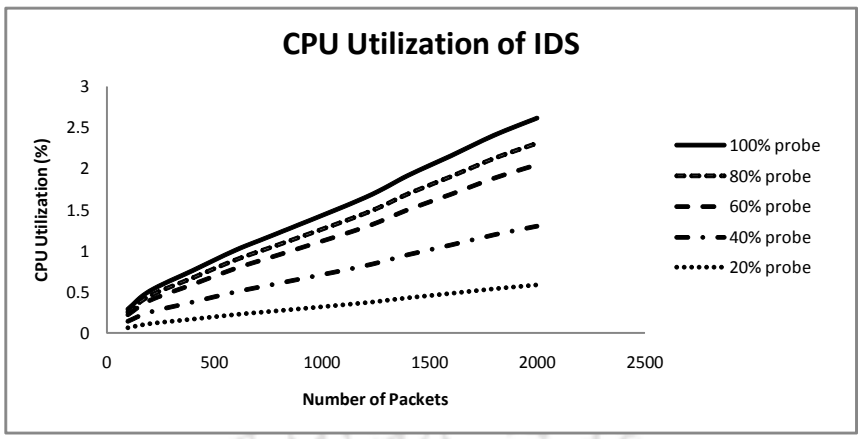

Figure 3.13: CPU Utilization of IDS

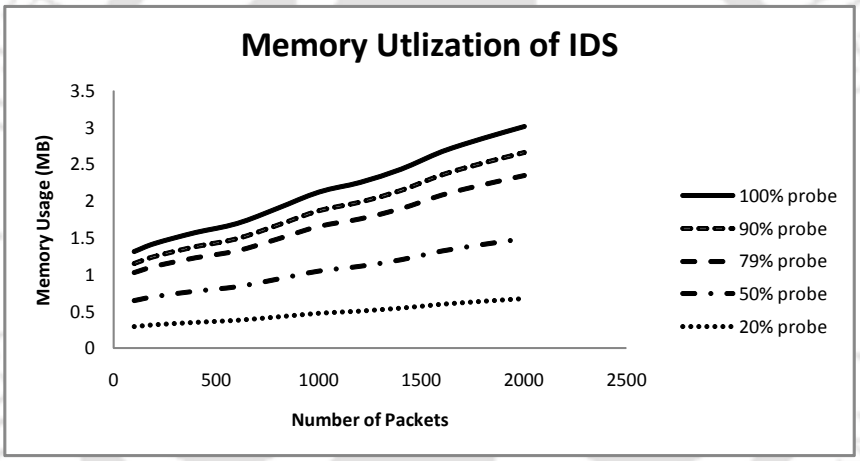

Figure 3.14: Memory Utilization of IDS

show CPU, memory and bandwidth utilization, respectively when running our IDS in Intel Core-2-Duo machine (with Ubuntu 9.10), under different amount of probing.

It may be noted that in the worst case of our experiment (i.e., maximum number of attack packets injected), CPU utilization is around 3%, total memory utilization by the system is about 1 GB and bandwidth utilization is around 2Mbps. So, we can conclude that under high number of attack packets also the IDS works without any problem in terms of resources.

3.6 Conclusion

In this chapter, an IDS using I-diagnosis based DES framework has been proposed for detecting ICMP Network/Host Unreachable message related attacks. The scheme employs an active mechanism to detect attacks without using any computationally expensive cryptographic mechanism. Also the scheme does not require change in protocol stack and does not violate layering architecture.

The proposed IDS allows detection of attack by diagnosing only in those paths where a fault is followed by an indicator event. This enables ICMP attack detection in spite of issues like network congestion. The state explosion issue in I-Diagnosability framework is addressed by augmenting the framework with model variables. A reduced *I*-detector (called *RI*-detector) has been developed by eliminating the redundant states and it has been shown that attack detection capability is not compromised.

ICMP attacks can be detected by separating normal and attack event sequence, by diagnosing in certain paths using I-diagnosis framework of DES. But in case of certain attacks like Low Rate TCP DoS attack, separation of normal and attack events are not possible by I-diagnosis or active DES framework because the attack and genuine traces may not clearly differ, but may differ with some probability. This motivated us to explore possibilities of attack detection of this type by sequence of events along with their probability of occurrence. In the next chapter we consider Induced Low Rate TCP DoS attack and adopted Stochastic DES framework [36] for its detection.

Chapter 4

Stochastic DES Based Detection and Mitigation of Induced Low rate TCP-targeted DoS attack

4.1 Introduction

Transmission Control Protocol (TCP) is a transport layer protocol which provides reliable and ordered delivery of stream of bytes from a computer to another computer [60]. In addition, TCP provides flow control, congestion avoidance and error control. Applications that need connection oriented reliable service use TCP, e.g., http, FTP, SMTP etc. Designed by DARPA in 1970's, the main emphasis on TCP is to provide reliable process to process service, dealing with congestion and errors in highly heterogeneous networks. However, almost no consideration is given that attackers would exploit this protocol to launch various attacks. Several attacks namely, SYN flood attack, TCP session hijacking, RST and FIN attack, attacks exploiting congestion avoidance algorithm etc. are prominent in TCP [60].

Among all the prevalent attacks in Transport Layer, Induced Low rate TCP attack is one of the new and challenging attacks as it can cause degradation of service to a large extent for which no proper solution is available. This motivated us to take up this attack and propose a solution for the same.

Broadly speaking, in TCP, congestion avoidance works by the following philosophy "if

a TCP segment sent before is acknowledged, increase the rate of transfer, else reduce the rate of transfer and retransmit". A segment may be lost (i.e., no acknowledgement) due to congestion or link failure or node failure, but it is observed that in wired network segment loss due to faults is much less than 1% [61]. Therefore whenever a segment is lost, it is assumed that there is congestion in the network. For every segment transmitted a timer is kept and if acknowledgement does not come within Retransmission Time Out (RTO) period then it is assumed that the segment is lost due to congestion and transfer rate is reduced.

In TCP, the rate at which a sender (server, say) transmits, depends on two factors –(i) state of the network in terms congestion and (ii) rate at which receiver (client, say) can accept data without being overloaded. According to RFC 2581 [8], which deals with congestion control in TCP, at the beginning, as state of the network is not known, the sender sends the segments using a "slow start phase". In the slow start phase, congestion window (cwnd) is set at 1 MSS (Maximum Segment Size) and 1 segment of 1 MSS is sent. When an acknowledgement for the segment is received (from the receiver), cwnd is doubled and two segments of 1MSS are sent. Although this phase is called slow start, but actually cwnd (and number of segments sent) increases exponentially with every round of transmission. Slow start is continued till either there is segment loss (i.e., acknowledgement does not arrive within RTO period) or cwnd reaches a threshold value. When cwnd reaches threshold value, cwnd is increased slowly. In case of loss of a segment, slow start phase is started again and lost segment is retransmitted. This reduces the rate of transmission drastically but is necessary because during congestion, time should be given to the network to recover.

This chapter deals with induced low rate TCP attack which works by exploiting the congestion avoidance algorithm of TCP protocol. An algorithm is proposed which will not only detect this attack but can mitigate this attack. We have adapted Stochastic Discrete Event System (DES) framework [36] for detecting Induced Low rate TCP attack. A DES is characterized by a discrete state space and some event driven dynamics [19]. In stochastic DES, probabilities of occurrence of events are also specified. The basic idea is to develop a stochastic DES model for the system under normal condition and also under each of the faulty (or attack) conditions. Following that, a stochastic state estimator called stochastic diagnoser is designed. Henceforth, the term diagnoser means stochastic diagnoser in this chapter. This diagnoser sevres two proposes, (i) diagnosability analysis–where certain

conditions on the diagnoser are checked off line to determine if failure can be diagnosed within finite time of its occurrence; (ii) on-line fault diagnosis–if fault is diagnosable, then the diagnoser is executed concurrently with the system. The diagnoser observes sequence of events generated by the system and raises an alarm if the sequence deviates from the normal behavior.

The chapter is organized as follows. Section 4.2 describes the attack mechanism of Induced Low rate TCP-targeted Denial of Service(Dos) attack and its existing countermeasures. Section 4.3 describes the Adopted Stochastic DES framework [36], augmented with model variables. In Section 4.4 the adapted DES framework is applied to model Induced Low rate TCP attack and its detection is exhibited. Experimental results are discussed in Section 4.5. The chapter is finally concluded in Section 4.6.

4.2 Induced Low rate TCP-targeted DoS attack and countermeasures

Low rate TCP attack, also called shrew attack [28] exploits the congestion avoidance algorithm. As the name suggests, attacker sends busty traffic for certain period of time and then keeps quiet and repeats this periodically. When the attacker congests the network to the target server, other genuine TCP flows destined to that server loose all or few of their segments. This causes genuine TCP flows to slow down their rate of transmission drastically. It may be noted that this attack keeps the average rate of sending busty traffic low, so that it cannot be identified by attack detection schemes like fair queuing [29] etc. Recently, V. Anil Kumar et al. [62]. have proposed a variant of low rate TCP attack termed as Induced Low Rate TCP attack. Here the attacker cleverly induces the server to perform low rate TCP attack instead of itself sending periodic busty traffic. Attacker achieves this by using a technique called optimistic acknowledgement [63]. As the name suggests, optimistic acknowledgement is a case when receiver sends a TCP acknowledgement for a segment that it has not received yet. The attack is started by downloading a large file from server. The attacker sends optimistic acknowledgements for a burst period for segments it is yet to receive. It may be noted that all segments except the last one in an entire TCP connection is of 1 MSS. So the attacker can easily guess the acknowledgment number for a

future segment (as it would be multiple of 1 MSS). Also, as the file is large, data transfer would take place for a reasonable amount of time, which can be utilized by the attacker to repeatedly send optimistic acknowledgements after a certain interval. By acknowledging segments before they are delivered, TCP wrongly believes that all segments are properly delivered and so the network to the receiver is not having any congestion. Rate of sending segments is increased by server to maximum limit, leading to congestion. In this way attacker induces server to perform the attack instead of itself being explicitly sending high amount of traffic. The attack is explained using a simple example shown in Figure 4.1.

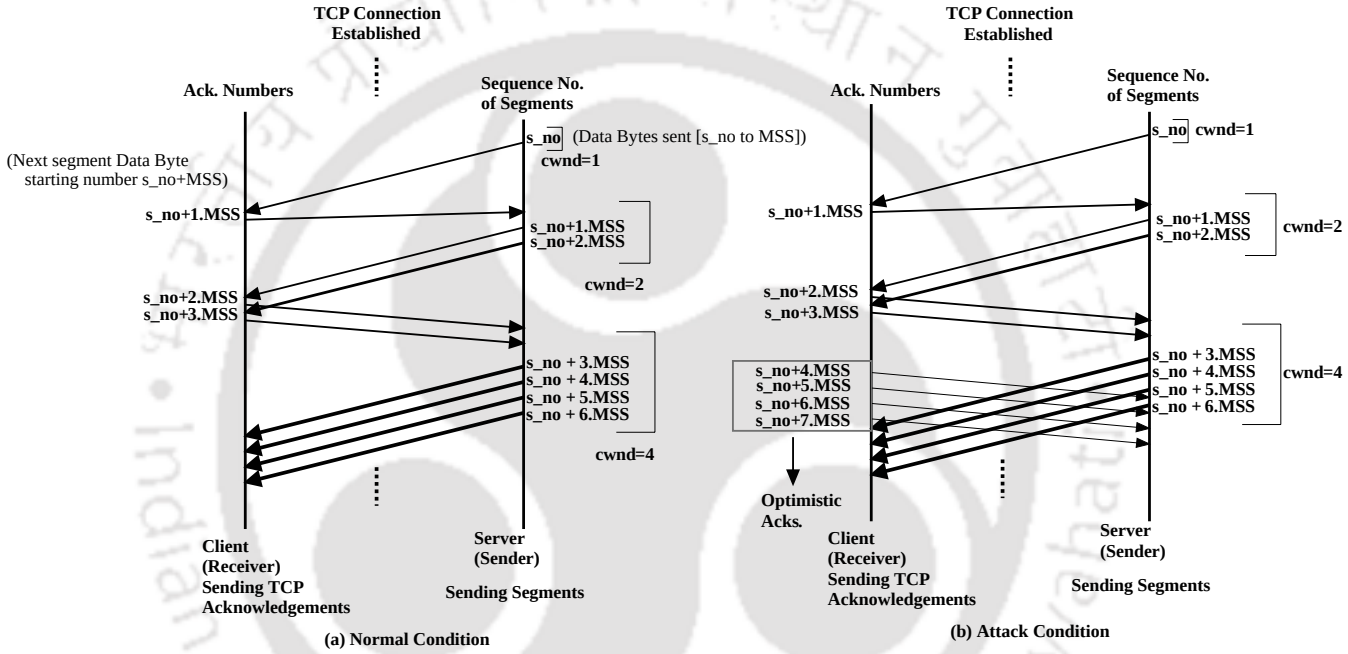

Figure 4.1: Simple example of Induced Low Rate TCP attack

In Figure 4.1 (a), sequence of TCP segments sent by server and their acknowledgements sent by receiver (after establishment of the TCP connection) are shown. The sequence numbers of the segments sent and the acknowledgement numbers of the acknowledgements received are also shown. Let the first segment have sequence number s_no, representing the starting of data bytes being sent (in that segment). In the first segment 1 MSS data bytes are being sent starting from s_no to $s\_no + 1 \times MSS - 1$. The corresponding acknowledgment has acknowledgement number as $s\_no + 1 \times MSS$, which is the starting data byte expected in next segment (i.e., acknowledge number is 1 more than $s\_no + 1 \times MSS - 1$). The acknowledgement with number $s\_no + 1 \times MSS$ is sent after segment having sequence number s_no is received. After this acknowledgement is received by the server it doubles

cwnd. cwnd is 2 MSS and 2 segment are sent thereby making sequence numbers as $s\_no + 1 \times MSS$ and $s\_no + 2 \times MSS$, respectively. When the acknowledgements for these segments are received, cwnd is doubled and 4 segments are sent.
Figure 4.1(b) shows the sequence of TCP segments sent by server and their acknowledgements when the receiver is launching Induced Low rate TCP attack. In this case, after sending a genuine acknowledgement having number $s\_no+2\times MSS$, it sends four optimistic acknowledgements having numbers $s\_no + 4 \times MSS$, $s\_no + 5 \times MSS$, $s\_no + 6 \times MSS$ and $s\_no + 7 \times MSS$. The attacker studies the sequence numbers of last few segments and finds that cwnd has doubled. So attacker expects that cwnd will keep doubling for the next few segments and sends optimistic acknowledgements. By acknowledging segments before they are delivered, TCP wrongly believes that all segments are properly delivered and so the network to the receiver is not having any congestion. Rate of sending segments is increased by server to maximum limit, leading to congestion.

Low rate TCP attack [28] being discovered about 10 years back, several solutions for its detection and prevention have been proposed. Use of fair queue scheduling has been proposed in [30]. Fair queue based schedules provide equal chances to segments of each TCP connection thereby avoiding overloading any connection by attacker. Anomaly based schemes which involve pattern matching of TCP packets with learnt attack behavior have been proposed in as series of works in [32, 31, 33, 34]. As Induced low rate TCP attack [62] has come up recently, there are few schemes for its mitigation, which work by detecting and blocking optimistic acknowledgments [63, 64]. A few works attempted to handle induced low rate TCP attack by executing the solutions of low rate TCP attack in the receiver. The major issue with these solution include lack of scalability, change in TCP header, lack of formal frameworks etc. In this chapter, we have adapted Stochastic Discrete Event System (DES) framework [36] for detecting Induced Low rate TCP attack, which addresses most of these issues.

4.3 Stochastic DES

In this section we present the Stochastic DES framework of [36] augmenting with *model variables*, which makes it applicable for detecting induced low rate TCP attack.

The *stochastic discrete event system (DES) model* G is defined as $G = \langle X, X_0, \Sigma, \Im, V \rangle$,

where

- X is the finite set of states.
- X_0 is the initial state.
- Σ is the set of events. Note that an event can be measurable or unmeasurable.
- $\mathfrak{I}$ is the set of transitions.
- V is the set of model variables.
  Each element v_i of V can take values from a domain Dom_i.

A transition $\tau \in \mathfrak{I}$ is defined as a six-valued tuple $\langle x, x^+, \sigma, check(v), assign(v), p \rangle$, where

- x is the initial state of the transition denoted as $initial(\tau)$.
- x^+ is the final state of the transition denoted as $final(\tau)$.
- $\sigma \in \{\Sigma \cup \epsilon\}$ is an event on which the transition τ is fired.
- $check(V)$ represents conditions on a subset of model variables. For firing τ, along with the enabling event σ, $check(V)$ should hold true.
- $assign(V)$ represents a subset of model variables and assignment of values from their corresponding domain, when τ fires.
- $p \in \mathbb{R}$ is a positive real number, which is the probability for τ to occur (when event is σ and $check(V)$ holds) given the system is in state x. p of τ is denoted as $prob(\tau)$. In other words, $prob(\tau) = prob(x^+|\sigma, check(V),\ x)$.

A *trace* of the model G is a sequence of transitions generated by G denoted as $s = \langle \tau_1, \tau_2, \cdots, \tau_f \rangle$, where $initial(\tau_{i+1}) = final(\tau_i)$, for $i = 1\ \ to\ \ (f-1)$. We denote $initial(\tau_1)$ as $initial(s)$ and $final(\tau_f)$ as $final(s)$. The set of all traces *generated* by G is the language of G, denoted as $L(G)$. Naturally, $L(G)$ is a subset of $\mathfrak{I}^w$, where $\mathfrak{I}^w$ is the set of all infinite sequences of $\mathfrak{I}$. Any finite member of $L(G)$ is in $\mathfrak{I}^*$, the Kleene closure of $\mathfrak{I}$. The post language of G after a trace s denoted as $L(G)/s = \{t \in \mathfrak{I}^* \mid st \in L(G)\}$. The probability of a trace $s = \langle \tau_1, \tau_2, \cdots, \tau_f \rangle$ can be calculated as $prob(\tau_1) \times prob(\tau_1) \cdots prob(\tau_f)$.
We assume the following properties.

- Property 1: All the states in the model G are reachable.
- Property 2: G is live, i.e., from each state $x \in X$, $\exists \tau \in \Im$ where $inital(\tau) = x$ such that $prob(\tau) > 0$.
- Property 3: The sum of probabilities of all transitions emanating from any state $x \in X$ is 1 i.e., $\sum_{initial(\tau)=x} prob(\tau) = 1$.

4.3.1 Model under Measurability

As mentioned, transition from a state in the model occur on enabling of an event and holding of $check(V)$. It may be noted that events may or may not be measurable. For example, event corresponding to "launching of an attack" is not measurable while the event for "receipt of a TCP connection request" is measurable. However, model variables are stored in the memory of the system (i.e., server), so their values are always measurable. Hence, the notion of measurability is associated only with the events (i.e., Σ) of a transition and not with model variables. The event can be either classified as measurable or unmeasurable and the event set Σ is divided as $\Sigma = \Sigma_m \cup \Sigma_{um}$, where Σ_m is the set of measurable events and Σ_{um} is a set of unmeasurable events. Given such a partition of events, the transitions can also be partitioned into two sets, $\Im_m$ and $\Im_{un}$, of *measurable* and *unmeasurable* transitions, respectively, based on the corresponding events being measurable or unmeasurable. Measurable transitions can be equivalent defined as follows:

Definition 4.1 (Measurement Equivalent Transitions and States): *Two transitions* $\tau_1 = \langle x_1, x_1^+, \sigma_1, check_1(V), assign_1(V), p_1 \rangle$ *and* $\tau_2 = \langle x_2, x_2^+, \sigma_2, check_2(V), assign_2(V), p_2 \rangle$ *are measurement equivalent if* $\sigma_1 = \sigma_2$ *(same event),* $check_1(V) \equiv check_2(V)$ *(same check over the same subset of variables in V) and* $assign_1(V) \equiv assign_2(V)$ *(same subset of model variables with same assignment).*
If $\tau_1 \equiv \tau_2$ *then the source states of the transitions are equivalent and so are the destination states, i.e.,* $x_1 \equiv x_2$ *and* $x_1^+ \equiv x_2^+$.

Definition 4.2 A **(projection operator)** $P : \Im^* \rightarrow \Im_m^*$ *can now be defined in the following manner.*
$P(\epsilon) = \epsilon$ *the null string,*

$P(\tau) = \tau$ *if* $\tau \in \Im_m$,
$P(\tau) = \epsilon$ *if* $\tau \in \Im_u$, $P(s\tau) = P(s)P(\tau)$, *where* $s \in L(G), \tau \in \Im$.
$P(s)$ *is termed as the* measurable trace *corresponding to the trace* s.

Definition 4.3 (Measurement equivalence of traces): *Two traces* s *and* s' *are equivalent, denoted as* sEs', *if* $P(s) = \langle \tau_1, \tau_2, ...\tau_f \rangle$, $P(s') = \langle \tau'_1, \tau'_2, ...\tau'_f \rangle$ *and* $\forall i (1 \leq i \leq f), \tau_i E \tau'_i$.

We loosely use the same symbol E to represent all (equivalence) relations.

4.3.2 Failure Modeling

Each state x is assigned a failure label defined by a function $C : x \rightarrow \{N\} \cup 2^{\{F_1, F_2, \cdots, F_k\}}$; $F_i, 1 \leq i \leq k$, stands for *permanent* failure status and N stands for normal status.

Definition 4.4 (Normal G-state): *A* G*-state* x *is normal if* $C(x) = \{N\}$. *The set of all normal states is denoted as* X_N.

Definition 4.5 (F_i-G-state): *A* G*-state* x *is failure state, or synonymously, an* F_i*-state, if* $F_i \in C(x)$. *The set of all* F_i*-states is denoted as* X_{F_i}.

Definition 4.6 (Normal-G-transition and trace): *A* G*-transition* τ *is called a normal* G*-transition if* $x, x^+ \in X_N$, *where* $x = initial(\tau)$ *and* $x^+ = final(\tau)$.
A G*-trace* s *is called normal-*G*-trace is all transitions in* s *are normal-*G*-transitions.*

Definition 4.7 (F_i-G-transition and trace): *A* G*-transition* τ *is called an* F_i*-*G*-transition if* $x, x^+ \in X_{F_i}$, *where* $x = initial(\tau)$ *and* $x^+ = final(\tau)$.
A G*-trace* s *is called* F_i*-*G*-trace is all transitions in* s *are* F_i*-*G*-transitions.*

A transition $\tau = \langle x, x^+, \sigma, check(V), assign(V), p \rangle$, where $C(x) \neq C(x^+)$, is called a *failure* transition indicating the first occurrence of some failure in the set $C(x^+) - C(x)$. For example, if a transition $\tau = \langle x, x^+, \sigma, check(V), assign(V), p \rangle$ occurs where $C(x) = \{N\}$ and $C(x^+) = \{F_i\}$ then it implies that fault F_i has occurred. A failure F_i causing transition is represented as τ_{F_i} and the set of all such transitions are represented as $\Im_{F_i}$.

The objective of failure diagnosis problem is to determine if failure F_i has occurred and it will be trivial if σ corresponding to τ_{F_i} is measurable. So for such failure causing

transitions, σ is unmeasurable, there is no check or assignment for model variables and p is some positive value. So the failure causing transitions are unmeasurable. In case of network attacks, fault is considered as attack (and obviously it occurrence is not directly measurable). So in this chapter attack and fault are used interchangeably. Since failures are assumed to be *permanent*, there is no transition from any state in x_{F_i} to any state in x_N. The event related to causing F_i will be denoted as σ_{F_i}.

The following definition, formalizes this notion of diagnosability of F_i for a stochastic DES.

Let $\Psi(S_{F_i}) = \{s \in L(G) | \tau = final(s) \in \Im_m \text{ and } final(\tau) \in X_{F_i}\}$. In words, $\Psi(S_{F_i})$ contains all traces which land in an F_i state with an measurable transition.

Definition 4.8 (F_i- Diagnosability for a stochastic DES model G): *The model G is said to be F_i-diagnosable for a fault F_i under a measurement limitation if the following holds*

$\forall Th(0 \leq Th \leq 1)[\exists n_{F_i} \in \mathbb{N}\{\forall s \in \Psi(X_{F_i}) \Rightarrow D\}]$

where condition D is $[(t \in L(G)/s \;\; |t| \geq n_{F_i} \;\; s.t. \;\; \exists y \in P^{-1}P(st) \wedge final(y) \notin X_{F_i}) \rightarrow (prob(st) < Th)]$

The above definition means the following. Let s be any finite prefix of a trace of G that ends in an F_i-state and let t be any sufficiently long continuation of s. Condition D then requires that for any threshold $0 \leq Th \leq 1$ there is a positive integer n_{F_i} such that if there is a sequence of transitions y, which is measurement equivalent with st and does not end into an F_i-state, then probability of occurrence of st is less than Th. In other words, traces of type st are responsible for non-diagnosability because st is measurement equivalent with some non-F_i G-trace y. However, the probability of such traces st can be brought below any threshold by extending t, thereby leading to fault diagnosis.

Now an example given in Figure 4.2 will be used to illustrate the theory. In the figure, states x_0, x_1 are N-states while x_0', x_1', x_2', x_3' are F_i-states. [1] The N and F_i transitions are shown in the figure as τ_i's and τ_i''s, respectively. τ_{1F_i} and τ_{2F_i} are failure causing transitions. In the figure the transitions are not explicitly detailed as their six tuples, however, corresponding p values are shown; for example τ_1 and τ_{1F_i} has a $p = 0.9$ and $p = 0.1$, respectively. This indicates that if system is in state x_0, then probability of occurrence of τ_1 is 0.9 and

[1] In this chapter, states (and transitions) with primes will be used to denote faulty states (and transitions), while the ones without primes are normal states (and transitions). If there are more than one faults (say F_i and F_j) then states (and transitions) corresponding to F_i will be denoted by single primes and those corresponding to F_k will be denoted by double primes.

occurrence of failure is 0.1. Here, $\tau_{1F_i}, \tau_{2F_i} \in \Im_{um}$. It is assumed that all other transitions (by virtue of their events) are measurable. From equivalence point of view, let $\tau_1 E \tau_1'$, $\tau_2 E \tau_2'$, $\tau_3 E \tau_3'$; this implies that $x_0 E x_0'$ and $x_1 E x_1'$.

The fault diagnosis problem is to determine if fault F_i has occurred within finite number n_{F_i} (where $n_{F_i} \in \mathbb{N}$) say, of transitions after occurrence of τ_{1F_i} or τ_{2F_i}. Let us consider a trace $s = \langle \tau_1, \tau_2, \tau_{1F_i}, \tau_1' \rangle$; obviously $s \in \psi(S_{F_i})$. For diagnosing F_i any sufficiently long but finite extension t of s of length n_{F_i} must ascertain that fault has occurred. In this case, if we extend $s = \langle \tau_1, \tau_2, \tau_{1F_i}, \tau_1' \rangle$ as $t = \langle (\tau_2', \tau_1')^k \rangle$, where k is arbitrarily large, we get $\exists y = \langle \tau_1, \tau_2, \tau_1, (\tau_2, \tau_1)^k \rangle \in P^{-1}(P(st)) \wedge final(y) = x_1 \notin X_{F_i}$. However, probability of $t = \langle (\tau_2', \tau_1')^k \rangle$ is $(0.1 \times 1)^k$ which reduces with increase in k. Therefore, given any threshold Th there is some $k = n_{F_i}$ such that if the length of t is k, probability of st is lower than Th. s can also be extended as $t = \langle (\tau_3')^k \rangle$. Here, $\exists y = \langle \tau_1, \tau_2, \tau_1, (\tau_3)^k \rangle \in P^{-1}(P(st)) \wedge final(y) = x_1 \notin X_{F_i}$. However, probability of $t = \langle (\tau_3')^k \rangle$ is $(0.4)^k$ which reduces with increase in k. Similar properties can be shown for all extension traces involving combinations of transitions $\tau_1', \tau_2', \tau_3'$ as extensions, e.g., $\langle \tau_3', (\tau_2', \tau_1')^k \rangle$, $\langle (\tau_3')^k, \tau_2', \tau_1' \rangle$ etc. In this example there are other ways of extending s, namely, $t1 = \langle \tau_2', \tau_1', \tau_2', \cdots, \tau_4' \rangle$, $t2 = \langle \tau_2', \tau_1', \tau_2', \cdots, \tau_4', \tau_5' \rangle$ and $t3 = \langle \tau_2', \tau_1', \tau_2', \cdots, \tau_4', \tau_5', (\tau_6')^k \rangle$ $(k \geq 1)$. It may be noted that $\nexists y \in P^{-1}(P(st1))$ such that $final(y) \notin X_{F_i}$; similar for $t2, t3$. Similar property holds for any trace s in the example. So fault is diagnosable.

In other words, trace $\langle \tau_1', \tau_2' \rangle$ is equivalent to $\langle \tau_1, \tau_2 \rangle$ (normal condition). So $\langle \tau_1', \tau_2' \rangle$ cannot detect the fault. Similar case is for trace $\langle \tau_1', \tau_3' \rangle$ as it is equivalent to $\langle \tau_1, \tau_3 \rangle$. Transition τ_4' causes the difference between normal and faulty condition and is mandatory for fault detection. After fault, probability of occurrence of τ_4' increases with time. So probability of continuing the loop $t = \langle \tau_1', \tau_2', \cdots \tau_1', \tau_2' \rangle$ or $\langle \tau_1', \tau_3', \tau_3', \cdots \tau_3' \rangle$ etc. decrease with time. Eventually, probability of traces of type st fall below any Th, thereby leading to fault detection.

In the next section, we discuss the procedure to construct the stochastic diagnoser from the model and the condition to be checked for diagnosability analysis.

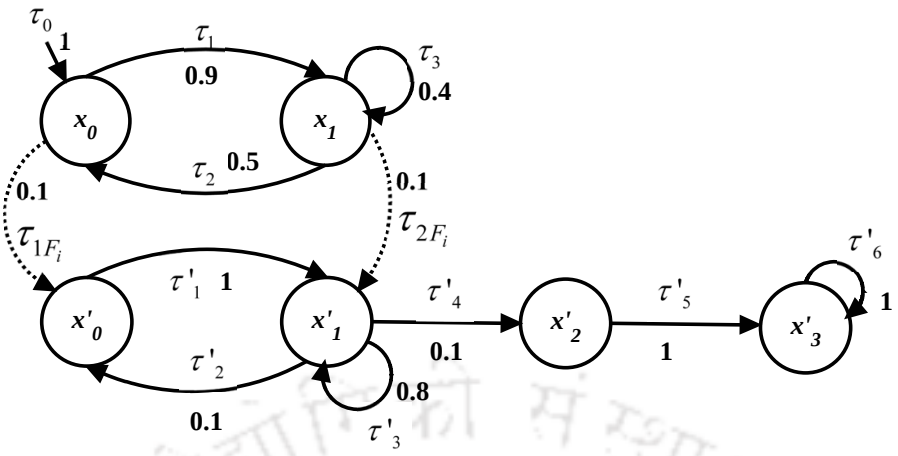

Figure 4.2: Simple example of stochastic DES

4.3.3 Stochastic Diagnoser

The stochastic diagnoser is represented as a directed graph $D = \langle Z, A \rangle$, where Z is the set of diagnoser states, called *D-states*, and A is the set of diagnoser transitions, called *D-transitions*. Henceforth, terms like transitions, states etc. of the DES are termed as model transitions, model states etc. to differentiate them from those of the diagnoser. Each D-state $z \in Z$ is an ordered set comprising a subset of equivalent model states representing the uncertainty about the actual state and each D-transition $a \in A$ is a set of equivalent model transitions representing the uncertainty about the actual transition that occurs. $initial(a)$ ($final(a)$) represents the source (destination) D-state of a. Also, with any D-transition a, a probability matrix Φ_a is associated. Φ_a represents probability of transitions from model states in $initial(a)$ to model states in $final(a)$; if $initial(a)$ has i model states and $final(a)$ has j model states, Φ_a is of dimension $i \times j$ (i rows and j columns).

Definition 4.9 Unmeasurable successor *of a D-state* z *is defined as* $\mathcal{U}(z) = \bigcup_{x \in z} \{x^+ | \tau = \langle x, x^+ \rangle \in \mathfrak{I}_{uo}\}$.

The unmeasurable reach *of* z *is the transitive closure (Kleene closure) of unmeasurable successors of* z *and is denoted as* $\mathcal{U}^*(z)$.

The diagnoser is constructed starting from the initial state(s) of the system model G. The initial D-state (z_0) is obtained first as $\mathcal{U}^*(X_0)$, where X_0 is the set of initial states of G. Given any D-state z, the transitions emanating from z are obtained as follows. Let $\mathfrak{I}_{mz}$ denote the set of measurable transitions defined from z. Let A_z be the set of all equivalence classes

of $\Im_{mz}$ under E. For each $a \in A_z$, a successor D-state z^+ of z such that $z^+ = final(a)$ can be created in the following way. Let $z_a^+ = \{final(\tau) | \tau \in a\}$; then $z^+ = \mathcal{U}^*(z_a^+)$ and $a = \langle z, z^+ \rangle$. So z^+ comprises system states which can be reached from system states in z via measurable equivalent transitions which are comprised in a. Further, new system states are added to z^+ which can be reached from the already existing system states in z^+ using unmeasurable transition sequence. The transition set of the diagnoser is augmented as $A \leftarrow A \cup \{a\}$ and the state set is augmented as $Z \leftarrow Z \cup \{z^+\}$.

To any D-transition $a = \langle z, z^+ \rangle$ a probability Φ_a is associated, whose values are determined as follows. For a system state $x_l \in z$ and another state $x_m \in z^+$, the probability of the transition $\tau \in a$ (from x_l to x_m in the model G) is represented by the l, m element of Φ_a and denoted as $\Phi_a[l, m]$. However, it may be possible that there is no direct transition from a model state $x_l \in z$ to a model state $x_m \in z^+$. In this case $\Phi_a[l, m] = 0$ if the clause holds: "there is no trace $\langle \tau_1, \tau_2, \cdots \tau_k \rangle$ where $initial(\tau_1) = x_l$ and $final(\tau_k) = x_m$, such that $\tau_1 \in \Im_m \in a$ and $\tau_2, \cdots \tau_k \in \Im_{um}$". Other wise if there are traces $s, s1 \cdots sp$ which satisfy the above clause, $\Phi_a[l, m] = prob(s) + prob(s1) \cdots prob(sp)$. It may be noted that in trace s (and also in $s1 \cdots sp$) only the first transition is measurable and comprised in a. The other transitions are unmeasurable.

It may be noted that the procedure for generating the probability matrix for the initial D-transition $a0$ (i.e., transition leading to the initial D-state z_0) is a bit different as there is no initial state for the initial D-transition. The dimension for Φ of $a0$, is $|z_0| \times 1$. It is assumed that probability of reaching the initial system state is 1. So $\Phi_{a0}[i] = 1$, if $x_i \in X_0$. Otherwise, if $x_i \in z_0$, but $x_i \notin z_0$ then, x_i is present in z_0 because of unmeasurable reach of some $x_j \in X_0 \in z_0$; in this case $\Phi_{a0}[i] = \sum_{\forall s \in \Im_{um}^*, initial(initial(s)) \in X_0 \wedge final(final(s)) = x_i} prob(s)$. In words, for the initial system state in z_0 the value in probability matrix is 1. For any other system state in z_0, the value in probability matrix is the sum of probability of traces from initial system state to the system state under consideration; as z_0 has the initial system state and its unmeasurable reach, so these traces comprise only unmeasurable transitions.

Construction of Stochastic Diagnoser is described in algorithm 4.1. The diagnoser for the stochastic DES model of Figure 4.2 is shown in Figure 4.3. Each D-state comprising a subset of measurement equivalent G-states and the set of measurement equivalent G-transitions corresponding to the D-transitions are illustrated in the figure. Some of the initial steps for this example are as follows.

Algorithm 4.1 Construction of Stochastic Diagnoser D for the Stochastic DES model G

Input: Stochastic DES model
Output: Stochastic Diagnoser.

Begin
$z_0 \leftarrow \mathcal{U}^*(X_0)$;
$Z \leftarrow z_0$;
$A \leftarrow a0$; /* initial transition (to z_1) */
if $x_i \in z_1 \in X_0$, $\Phi_{a0}[i] = 1$;/* construction of Φ for $a0$, whose dimension is $|z_1| \times 1$*/
else $\Phi_{a0}[i] = \sum_{\forall s \in \Im^*_{uo}, initial(initial(s)) \in X_0 \wedge final(final(s)) = x_i} prob(s)$
for all $z \in Z$ Do {
    /* Find the set of measurable G-transitions ($\Im_{mz}$) outgoing from z */
    $\Im_{mz} \leftarrow \{\tau | \tau \in \Im_m \wedge initial(\tau) \in z\}$;
    /* Find the set of all measurement equivalent classes A_z, of $\Im_{mz}$*/
    For all $a \in A_z$ {
        $z_a^+ = \{final(\tau) | \tau \in a\}$;
        $z^+ = \mathcal{U}^*(z_a^+)$;
        $Z = Z \cup \{z^+\}$;
        $A = A \cup \{a\}$;
        for all $x_i \in z$ Do {
            for all $x_j \in z^+$ Do {
            if $\exists \tau \in a$, such that $initial(\tau) = x_i$ and $final(\tau) = x_j$, $\Phi_a[i,j] = prob(\tau)$;
            else if, there exists traces $(s, s1, \cdots sp)$ of type $\langle \tau_1, \tau_2, \cdots \tau_k \rangle \in \Im^*$,
            such that $initial(\tau_1) = x_i$, $final(\tau_k) = x_j$ and $\tau_1 \in a \wedge \tau_2, \cdots \tau_k \in \Im_{um}$,
            $\Phi_a[i,j] = prob(s) + prob(s1) \cdots prob(sp)$;
            else $\Phi_a[i,j] = 0$;
            }
        }
    } /* For all $a \in A_z$ */
} /* for $z \in Z$ */
End

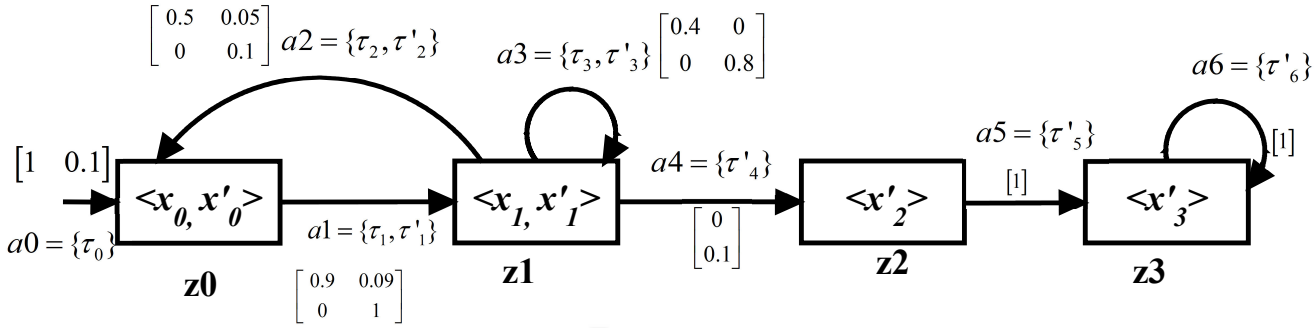

Figure 4.3: Stochastic diagnoser for DES of Figure 4.2

(i) The initial state of the diagnoser i.e., z_0 is obtained as follows. First $x_0 \in X_0$ is inserted in z_0. Now, all states in $\mathcal{U}^*(x_0)$ are inserted in z_0; x'_0 is within unmeasurable reach of x_0 and is inserted in z_0. So, $z_0 = \langle x_0, x'_0 \rangle$.

The probability matrix associated with initial D-transition $a0$ (whose destination is 2×1) is [1 0.1] which is obtained as follows. $\Phi_{a0}[1] = 1 = prob(\tau_0)$ as $x_0 (\in X_0) \in z_1$ and $final(\tau_0) = x_0$. $\Phi_{a0}[2] = prob(\tau_{1F_i}) = 0.1$; there exists a trace $s = \langle \tau_{1F_i} \rangle$ (where $\tau_{1F_i} \in \Im_{um}$) such that $initial(\tau_{1F_i}) = x_0$ and $final(\tau_{1F_i}) = x'_0$.

(ii) The outgoing O-transitions from z_0 are obtained as follows. Here, $\Im_{mz0} = \{\tau_1, \tau'_1\}$ which are all the outgoing measurable transitions from G-states in z_0. Now, $A_{z_1} = \{\{\tau_1, \tau'_1\}\}$ as $\tau_1 E \tau'_1$. Corresponding to $\{\tau_1, \tau'_1\}$ there is a D-transition $a1$.

The probability matrix associated with D-transition $a1$ is $\Phi_{a1} = \begin{bmatrix} 0.9 & 0.09 \\ 0 & 1 \end{bmatrix}$.

- $\Phi_{a1}[1,1] = 0.9$ because $prob(\tau_1) = 0.9$.
- $\Phi_{a1}[1,2] = 0.09$ because there is a trace $s = \langle \tau_1, \tau_{2F_i} \rangle$, where $initial(\tau_1) = x_0$, $final(\tau_{2F_i}) = x'_1$, $\tau_1 \in a1 \in \Im_m$ and $\tau_{2F_i} \in \Im_{um}$; so, $\Phi_{a1}[1,2] = prob(s) = prob(\tau_1) \times prob(\tau_{2F_i}) = 0.09$.
- $\Phi_{a1}[2,1] = 0$ because x'_0 cannot reach x_1 by any trace.
- $\Phi_{a1}[2,1] = 1$ because $prob(\tau'_1) = 1$.

(iii) The destination D-state corresponding to $a1$ is obtained as follows. $z1^+_{a1} = \{\langle x_1, x'_1 \rangle\}$ as $a1$ comprises D-transitions τ_1, τ'_1 and $final(\tau_1) = x_1$ and $final(\tau'_1) = x'_1$. Further, $z_1^+ = \{x_1, x'_1\}$ as $\mathcal{U}^*(\{x_1\}) = \{x_1, x'_1\}$ and $\mathcal{U}^*(\{x'_1\}) = \{x'_1\}$. Thus, the destination D-state of the D-transition $a1$ is $z_1 : \{\langle x_1, x'_1 \rangle\}$.

4.3.3.1 Diagnosability analysis

Before discussing the condition to be checked for diagnosability analysis on the diagnoser, certain definitions and properties are introduced.

Definition 4.10 (Embedding of G-traces in D-traces): *Given a D-trace $\gamma = \langle a1, a2, ..., ak\rangle$, a G-trace s, where $P(s) = \langle \tau_1, \tau_2..., \tau_k\rangle$, is said to be embedded in γ, if $\tau_i \in ai$, $1 \leq i \leq k$. The set of all G-traces embedded in a D-trace γ is represented as $A_D(\gamma)$.*

Property 4.1 *If two traces $t, y \in A_D(\gamma)$, where t is F_i-G-trace and y is non-F_i-G-trace, then the D-states traversed by γ are F_i-uncertain.*

The property also follows from the diagnoser construction. As any D-transition $a \in \gamma$ has an non-F_i-G-transition and a F_i-G-transition (which are equivalent), so source and destination D-states of a are F_i-uncertain.

The fault label of any D-state $z = \langle x_1, x_2, ..x_i, ...\rangle$ is defined as $C(z) = \bigcup_{x \in z} C(x)$.

Definition 4.11 (Normal D-state): *A D-state z is called* normal *and denoted as z_N, if $C(z) = \{N\}$; the set of all normal D-states is denoted as Z_N.*

Definition 4.12 (F_i-D-state): *A D-state z is called an F_i-D-state and denoted as z_{F_i}, if $F_i \in C(z)$. The set of all F_i D-states is denoted as Z_{F_i}.*

Definition 4.13 (F_i-certain D-state): *An F_i-D-state z is called an F_i-certain D-state if $z \subseteq X_{F_i}$.*

Definition 4.14 (F_i-uncertain D-state): *An F_i-D-state which is not F_i-certain is called F_i-uncertain.*

In words, F_i-certain D-state comprises only F_i-system states, while F_i-uncertain D-state comprises some F_i-system states and some non-F_i system states. In Figure 4.3, z_0, z_1 are F_i-uncertain states and z_2, z_3 are F_i-certain states. So, if the diagnoser reaches any F_i-certain D-state failure is diagnosed. By Property 4.1 if there is a D-trace γ which moves in F_i-uncertain D-states, there is a F_i-G-trace t which is embedded in γ. After failure, diagnoser moves in γ by virtue of t. However, as there is another non-F_i-G-trace y say, equivalent to t, fault cannot be diagnosed till γ is exited. If the probability of t decreases

with increase in length, the probability of continuing γ also decreases. If this holds for all G-traces which cause diagnoser to move in F_i-uncertain D-states, then fault is diagnosable because eventually some F_i-certain D-state is reached by the diagnoser. So, according to diagnosability (Definition 4.4) the following condition need to be checked in the diagnoser to ascertain diagnosability; this is called diagnosability analysis condition as defined below.

Definition 4.15 (F_i-diagnosability analysis condition): *Failure F_i is diagnosable if, after occurrence of F_i, probability of moving in F_i-uncertain D-states falls with increase in length of any F_i-G-trace.*

In the present example, we will show that probability of moving in z_0, z_1 (which are F_i-uncertain D-states) decreases with increase in length of system traces after failure. Let us consider the D-trace $\gamma = \langle a0, a1, a3, a3 \rangle$. If the diagnoser moves in γ, then either system trace is $\tau_0, \tau_1, \tau_3, \tau_3, \tau_3$ (normal condition) or $\tau_0, \tau_{1F_i}, \tau'_1, \tau'_3, \tau'_3, \tau'_3$ (fault condition) or $\tau_0, \tau_1, \tau_{2F_i}, \tau'_3, \tau'_3, \tau'_3$ (fault condition). The probability of the D-trace γ is matrix product of probability matrices associated with the D-transitions in the trace.

$$\begin{bmatrix} 1 & 0.1 \end{bmatrix} . \begin{bmatrix} 0.9 & 0.09 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} 0.4 & 0 \\ 0 & 0.8 \end{bmatrix} . \begin{bmatrix} 0.4 & 0 \\ 0 & 0.8 \end{bmatrix} \begin{bmatrix} 0.4 & 0 \\ 0 & 0.8 \end{bmatrix} = \begin{bmatrix} 0.058 & 0.098 \end{bmatrix}$$

The D-trace γ leads to z_1, where we have system states x_1, x'_1. The probability of γ is [0.058 0.098], which implies that probability of reaching system state x_1 and x'_1 is 0.058 and 0.098, respectively. So, if failure has occurred then probability of the diagnoser being in F_i-uncertain D-state z_1 (by virtue of system state being x'_1) is 0.098. In other words, after failure γ occurs due to F_i-G-trace $\tau'_1, \tau'_3, \tau'_3, \tau'_3$, whose probability is 0.098. Now if we extend the D-trace γ by another D-transition $a3$ (i.e., by extending the corresponding F_i-G-trace by τ'_3), then probability of the extended D-trace is [0.0232 0.0784]. So if failure has occurred, probability of the diagnoser being in z_1 is 0.0784 i.e., probability of the F_i-G-trace $\tau'_1, \tau'_3, \tau'_3, \tau'_3, \tau'_3$ is 0.0784. It can be shown that if γ gets extended as $a3^k$ ($k \geq 1$), probability of diagnoser being in z_1 will keep falling; this occurs because probability of extending the F_i-G-trace $\tau'_1, \tau'_3, \tau'_3, \tau'_3, \tau'_3$ with τ'_3 results in lowering its probability. So as per Definition 4.4, the probability of this F_i-G-trace under question, which results in non-diagnosability (as the corresponding D-trace γ moves in F_i-uncertain states), decreases with extension. Similar can be shown for other D-traces moving in F_i-uncertain D-states. For example, let $\gamma 1 = \langle a0, a1, a2, a1 \rangle$. The probability of $\gamma 1$ is [0.405 0.105], as computed below.

$$\begin{bmatrix} 1 & 0.1 \end{bmatrix}.\begin{bmatrix} 0.9 & 0.09 \\ 0 & 1 \end{bmatrix}.\begin{bmatrix} 0.5 & 0.05 \\ 0 & 0.1 \end{bmatrix}.\begin{bmatrix} 0.9 & 0.09 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.405 & 0.105 \end{bmatrix}$$

It can be checked, if $\gamma 1$ is extended as $a2, a1$, the probability is [0.203 0.031] (i.e., falls). Now we consider the trace $\gamma 2 = \langle a0, a1, a4, a5 \rangle$, which takes the diagnoser out of F_i-uncertain D-states, leading to z_3, resulting is fault diagnosis. The probability of $\gamma 2$ is [0.019], as computed below.

$$\begin{bmatrix} 1 & 0.1 \end{bmatrix}.\begin{bmatrix} 0.9 & 0.09 \\ 0 & 1 \end{bmatrix}.\begin{bmatrix} 0 \\ 0.1 \end{bmatrix}.\begin{bmatrix} 1 \end{bmatrix}.\begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} 0.019 \end{bmatrix}$$

It can be checked, if $\gamma 2$ is extended as $a6$ (corresponding F_i-G-trace τ_6'), the probability is [0.019] (i.e., does not fall will extension). It may be noted that after reaching F_i-certain D-state, the probability of remaining in F_i-certain states is constant (even with increase in length of the trace).

In next section we present the application of this adapted Stochastic DES framework for detection of induced low rate TCP attack.

4.4 Proposed scheme: Application of Stochastic DES for detecting Induced Low Rate TCP attack

The basic block diagram of the proposed scheme is shown in Figure 4.4. As shown in the figure, the receiver side (which downloads a large file from server) runs TCP without any modification. In the server side, TCP is augmented with a module "SEND_RCV_HANDLER()", which is responsible for generating random numbers, selecting segments where less data bytes are to be sent and accordingly reorganizing the data buffer. Also, the module generates events like received acknowledgement, sent segment, sent segment with reduced data bytes etc. along with values of associated parameters (modeled as model variables in DES) e.g., source IP of the segment, sequence number, acknowledgement number, number of bytes less than MSS (if the segment is sent with less data bytes) etc. Some of these values are in the TCP header of the segment and others are generated by the module. These events are given to the "DES diagnoser" module, which performs two tasks

(i) Diagnosability analysis: It is an of-line one time procedure which checks the Diagnosability analysis condition. If the condition is satisfied, then the diagnoser is implemented and used for attack detection; we will show later that the diagnoser designed to detecting

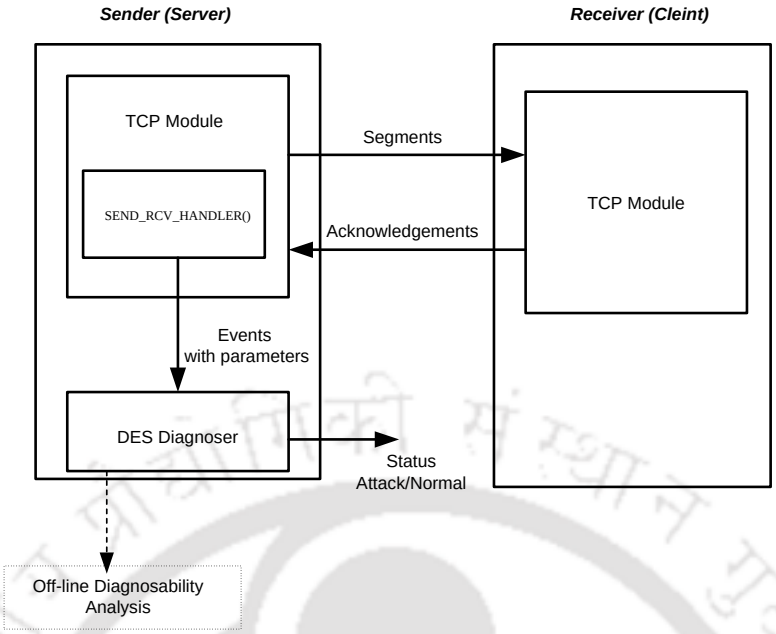

Figure 4.4: Block Diagram of the proposed scheme

the low rate TCP attack satisfies the diagnosability condition.
(ii) On-line attack detection: Monitoring the events and the parameters generated by "SEND_RCV_HANDLER()" module and if some F_i-certain state is reached, alarm is generated denoting attack detection.

Henceforth in the discussion, the following short notations are used:
IPS - Source IP Address; *IPD* - Destination IP Address; *PTS* - Source port number; *PTD* - Destination port number; *TSN* - TCP segment with 1 MSS data bytes; *TSL* - TCP segment with some data bytes less than MSS; *TA*-TCP acknowledgement; *SN*-sequence number; *AN*-acknowledgement number.
Source IP of *TSN* is denoted as TSN_{IPS}; similar type subscripting is used in this chapter e.g., TA_{AN} is the acknowledge number of the TCP acknowledgement *TA*.
The SEND_RCV_HANDLER() module is discussed in Algorithm 4.2.

The algorithms works as follows. It takes input as (i) TCP segments from the buffer which are yet to be sent (this is possible as SEND_RCV_HANDLER() is a part of the modified TCP); (ii) L1 and L2 are two fixed positive numbers; (iii) TCP acknowledgements; (iv) TEST_FLAG indicating when to detect attack by sending a segment with less data bytes. Periodically two random numbers RND1 and RND2 are generated within [1, L1]

Algorithm 4.2 SEND_RCV_HANDLER()

Input : TCP segments and TCP Acknowledgements, L1 and L2, TEST_FLAG;
Output: Events: TSN, TSL, TA;
Variables: RND1, RND2, SEG_NO=0;

```
Begin
while(TCP connection not terminated) {
      if (TEST_FLAG==1) {
      Generate random numbers RND1 between 1 and L1, and RND2 between 1 and L2;
      }
      if (TCP segment is not for retransmit) {
          SEG_NO= (SEG_NO%L1) +1;
          if((RND1==SEG_NO) AND (TEST_FLAG==1)) {
              Decrement RND2 number of bytes in the current TCP
              segment to be transmitted and adjust transmission buffer.
              Generate Event TSL;
              Send the TCP segment;
              }
          else if(RND1!=SEG_NO) {
              Generate Event TSN;
              Send the TCP segment;
              }
      }
      if (TCP segment an acknowledgment)
      Generate Event TA;
} End
```

and [1, L2] respectively. RND1 is used to select a segment where RND2 number of data bytes less than MSS is sent. The output of the module is events: TSN, TSL and TA which are passed to the DES diagnoser. The variables RND2 and TEST_FLAG are also model variables; so these variables are shared between diagnoser and the handler. TEST_FLAG is set by DES diagnoser indicating that attack detection is to be started by generating RND1 and RND2 and sending $RND1^{th}$ segment with RND2 data bytes less.

When a TCP connection is established, the DES diagnoser is executed which continues till the connection is terminated. The initial transition of the diagnoser makes TEST_FLAG=1. In the module, initially SEG_NO=0; SEG_NO represents the number of new segments that are transmitted. The module checks if TEST_FLAG=1 (which is true, as initially DES diagnoser sets it to 1) and generates two random numbers RND1 and RND2 between [1, L1]

and [1, L2] respectively. Following that, it is checked if the next segment to be transferred is not retransmission; if so, SEG_NO is incremented. It may be noted that increment is SEG_NO%L1 +1 (not SEG_NO+1) because randomly some segments are selected when SEG_NO becomes equal to RND1, less data bytes are sent. As RND1 is between [1, L1] we do not monotonically increase SEG_NO, rather truncate it at L1 and restart from 1. Next, it is checked if SEG_NO matches RND1 and TEST_FLAG=1; it implies that detection of attack is to be done using the current segment by sending RND2 data bytes less in it. Event TSL is passed to diagnoser and the TCP segment is sent. On receiving the event TSL, diagnoser makes TEST_FLAG=0 and it remains so until conclusion regarding normal/attack condition is made by diagnoser; this ensures that before making a conclusion with the current segment no new segment (with less data bytes) is sent for attack detection. If SEG_NO does not match RND1 event TSN is sent to diagnoser and the TCP segment is sent. If a segment is received which is an acknowledgement from the receiver event TA is passed to diagnoser. It may be noted that if a segment is for retransmission this module does not do anything as TCP takes care of it i.e., retransmission.

Now we discuss how the diagnoser detects the attack on-line. Following that we develop it using the formalism introduced in Section 4.3 and finally show that diagnosability condition is met.

Based on RND1, some TCP segment being sent is selected. In that segment the data bytes being sent is reduced by RND2; so the TCP segment now has some bytes less than MSS. In the normal condition, the corresponding acknowledgement will have the acknowledgement number as $s\_no + k \times MSS - RND2$ (where s_no is the sequence number of the first TCP segment after connection is established, k is a positive integer) i.e., not of the form "s_no + multiple of MSS". For example, if s_no=1001 and MSS=1000, then the acknowledgement number for the first segment is 2001 i.e., $s\_no + 1 \times MSS$. In the next segment let 20 bytes (RND2 is 20, say) less is being sent. So the acknowledgement number must be 2981 i.e., $1001 + 2 \times 1000 - 20$. This is modeled as the normal stochastic DES. Now, if an attacker is present it sends the acknowledgement before the segment is received. There may be two cases, (i) acknowledgement number as "s_no + multiple of MSS" is sent even for the segment with less data bytes, (ii) guesses RND1 and RND2 and sends acknowledgment with number $s\_no + k \times MSS - RND2$. These two facts are modeled as failure stochastic DES. From the DES model the diagnoser is constructed. In case (i)

diagnoser satisfies the diagnosability condition vacuously because there is no uncertain state. It may be noted that there is no uncertain state in this case faulty behavior (attack situation) is different from normal situation. However, in case (ii) there will be uncertain states in the diagnoser because of the assumption that attacker can correctly guess RND1 and RND2 thereby making faulty behavior same as that of attack. But, it may be noted that probability of guessing acknowledgment number correctly for the segment with less data bytes is $1/(RND1 \times RND2)$ and repeating the same k times (i.e., k different segments with less data bytes) is $(1/(RND1 \times RND2))^k$, which decreases with increase in k. This will imply that probability of moving in fault uncertain states by the diagnoser (i.e., virtue of case (ii)) will fall and finally by a wrong guess, diagnoser will move to a fault certain state. The stochastic DES model used to represent TCP send-acknowledgment sequence at normal condition and optimistic acknowledgment is shown in Figure 4.5.

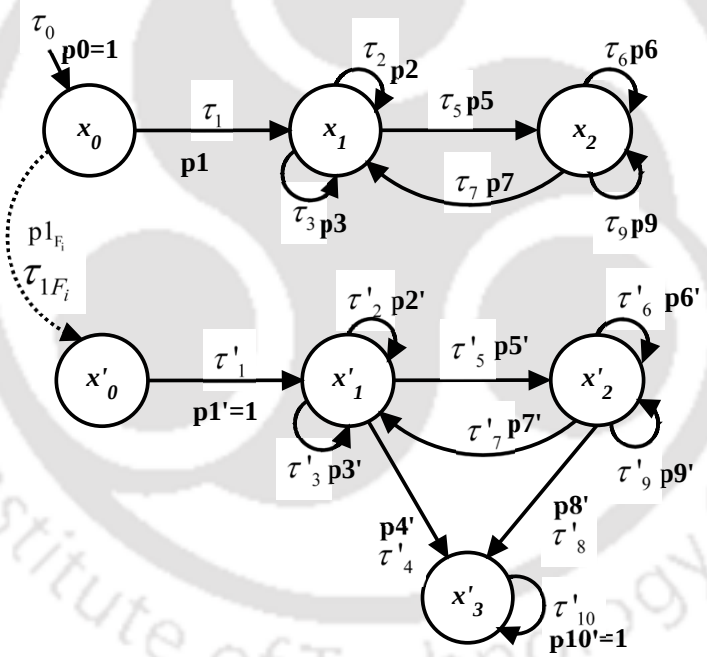

Figure 4.5: Stochastic DES Model of TCP send-acknowledgment sequence at normal and attack condition

The various components of the stochastic DES model G for TCP send-acknowledgment sequence are as follows

- $X = \{x_0, x_1, x_2, x'_0, x'_1, x'_2, x'_3\}$ is the finite set of states.
- $x_0 = X_0$ is the initial state.

- $\Sigma = \{TSN, TSL, TA, attack\}$ is the set of events; $\Sigma_m = \{TSN, TSL, TA\}$ and $\Sigma_{um} = \{attack\}$.
- $\Im$ is the set of transitions shown in Figure 4.5 and explained in Table 4.1.
- $V = \{ips, pts, ipd, ptd, seq\_no, seq\_no1, RND2, TEST\_FLAG\}$ is the set of model variables. *ips* and *ipd* have the domain as $\{d.d.d.d|d \in \{1, 2, \cdots, 255\}\}$. *pts* and *ptd* have the domain as 0 to $2^{16} - 1$; in TCP header, port field is of 16 bits. seq_no and seq_no1 have the domain as 0 to $2^{32} - 1$; in TCP header, sequence number field is of 32 bits. Domain of $RND2$ is between 1 and L2. TEST_FLAG is a Boolean variable.

Table 4.1: Transitions of the DES model of Figure 4.5

Tr.	Transition Description
τ_0	$\langle -, x_0, TRUE, -, TEST\_FLAG \leftarrow 1, 1\rangle$
τ_1	$\langle x_0, x_1, TSN, -, \{ips \leftarrow TSN_{IPS}, ipd \leftarrow TSN_{IPD}, pts \leftarrow TSN_{PTS}, ptd \leftarrow TSN_{PTD}, seq\_no \leftarrow TSN_{SN}\}, p1\rangle$
τ_2	$\langle x_1, x_1, TSN, \{ips = TSN_{IPS}, ipd = TSN_{IPD}, pts = TSN_{PTS}, ptd = TSN_{PTD}\}, seq\_no \leftarrow TSN_{SN}, p2\rangle$
τ_3	$\langle x_1, x_1, TA, \{ips = TA_{IPD}, ipd = TA_{IPS}, pts = TA_{PTD}, ptd = TA_{PTS}, TA_{AN} \leq (seq\_no + MSS)\}, -, p3\rangle$
τ_5	$\langle x_1, x_2, TSL, \{ips = TSN_{IPS}, ipd = TSN_{IPD}, pts = TSN_{PTS}, ptd = TSN_{PTD}\}, \{TEST\_FLAG \leftarrow 0, seq\_no1 \leftarrow TSL_{SN}\}, p5\rangle$
τ_6	$\langle x_2, x_2, TA, \{ips = TA_{IPD}, ipd = TA_{IPS}, pts = TA_{PTD}, ptd = TA_{PTS}, TA_{AN} \leq (seq\_no + MSS)\}, -, p6\rangle$
τ_7	$\langle x_2, x_1, TA, \{ips = TA_{IPD}, ipd = TA_{IPS}, pts = TA_{PTD}, ptd = TA_{PTS}, (TA_{AN} > (seq\_no + MSS))AND((TA_{AN} - seq\_no1 + RND2)\%MSS = 0)\}, TEST\_FLAG \leftarrow 1, p7\rangle$
τ_9	$\langle x_2, x_2, TSN, \{ips = TSN_{IPS}, ipd = TSN_{IPD}, pts = TSN_{PTS}, ptd = TSN_{PTD}\}, -, p_9\rangle$
τ_{1F_i}	$\langle x_0, x'_0, attack, -, -, p1_{F_i}\rangle$
τ'_1	$\langle x'_0, x'_1, TSN, -, \{ips \leftarrow TSN_{IPS}, ipd \leftarrow TSN_{IPD}, pts \leftarrow TSN_{PTS}, ptd \leftarrow TSN_{PTD}, seq\_no \leftarrow TSN_{SN}\}, p1'\rangle$
τ'_2	$\langle x'_1, x'_1, TSN, \{ips = TSN_{IPS}, ipd = TSN_{IPD}, pts = TSN_{PTS}, ptd = TSN_{PTD}\}, seq\_no \leftarrow TSN_{SN}, p2'\rangle$
τ'_3	$\langle x'_1, x'_1, TA, \{ips = TA_{IPD}, ipd = TA_{IPS}, pts = TA_{PTD}, ptd = TA_{PTS}, TA_{AN} \leq (seq\_no + MSS)\}, -, p3'\rangle$
τ'_4	$\langle x'_1, x'_3, TA, \{ips = TA_{IPD}, ipd = TA_{IPS}, pts = TA_{PTD}, ptd = TA_{PTS}, TA_{AN} > (seq\_no + MSS)\}, -, p4'\rangle$
τ'_5	$\langle x'_1, x'_2, TSL, \{ips = TSN_{IPS}, ipd = TSN_{IPD}, pts = TSN_{PTS}, ptd = TSN_{PTD}\}, \{TEST\_FLAG \leftarrow 0, seq\_no1 \leftarrow TSL_{SN}\}, p5'\rangle$
τ'_6	$\langle x'_2, x'_2, TA, \{ips = TA_{IPD}, ipd = TA_{IPS}, pts = TA_{PTD}, ptd = TA_{PTS}, TA_{AN} \leq (seq\_no + MSS)\}, -, p6'\rangle$
τ'_7	$\langle x'_2, x'_1, TA, \{ips = TA_{IPD}, ipd = TA_{IPS}, pts = TA_{PTD}, ptd = TA_{PTS}, (TA_{AN} > (seq\_no + MSS))AND((TA_{AN} - seq\_no1 + RND2)\%MSS = 0)\}, TEST\_FLAG \leftarrow 1, p7'\rangle$
τ'_8	$\langle x'_2, x'_3, TA, \{ips = TA_{IPD}, ipd = TA_{IPS}, pts = TA_{PTD}, ptd = TA_{PTS}, (TA_{AN} > (seq\_no + MSS))AND((TA_{AN} - seq\_no1 + RND2)\%MSS \neq 0)\}, -, p8'\rangle$
τ'_9	$\langle x'_2, x'_2, TSN, \{ips = TSN_{IPS}, ipd = TSN_{IPD}, pts = TSN_{PTS}, ptd = TSN_{PTD}\}, -, p9'\rangle$
τ'_{10}	$\langle x'_3, x'_3, TRUE, -, -, p10'\rangle$

An overview of the DES model for the normal/attack scenario is now explained.

- States x_0, x_1, x_2 and transitions $\tau_0, \tau_1, \tau_2, \tau_3, \tau_5, \tau_6, \tau_7, \tau_9$ represent the system under **normal condition**. As shown in Figure 4.5, τ_0 is the initial transition leading to initial state. In the modeling we assume that TCP connection has been established and then τ_0 takes place. We do not explicitly keep any event to trigger τ_0. For simplicity, we rather assume that when TCP connection is established τ_0 occurs (i.e., DES model

is invoked). As shown in Table 4.1, $initial(\tau_0) = -$, implying there is no initial state and $final(\tau_0) = x_0$. $\sigma = TRUE$ implies that the transition is always enabled i.e., when the model starts, automatically x_0 is reached. $check(V) = -$, implies that there is no condition to be checked on model variables i.e., condition on model variables is always satisfied for the transition. $assign(V) = \{TEST\_FALG \leftarrow 1\}$ implies that variable TEST_FALG is assigned value 1 and detection of attack can be started (by sending a random segment with less data bytes). Probability is 1.

- As we are modeling induced low rate TCP attack, we concentrate on segments being sent from the server (that correspond to file download by receiver) and the corresponding acknowledgements. So after the model is started and is at x_0 we look for the first TCP segment being sent; this is modeled using τ_1. From Table 4.1, $initial(\tau_1) = x_0$, $final(\tau_1) = x_1$. $\sigma = TSN$ implies that the transition is enabled when the SEND_RCV_HANDLER() generates the event TSN (after sending a new TCP segment with MSS data bytes). Here, $check(V) = -$. $assign(V) = \{ips \leftarrow TSN_{IPS}, ipd \leftarrow TSN_{IPD}, pts \leftarrow TSN_{PTS}, ptd \leftarrow TSN_{PTD}, seq\_no \leftarrow TSN_{SN}\}$. The parameters to identify the segments of a TCP connection are source IP, source port, destination IP and destination port. All these parameters corresponding to the TCP segment that has been sent are stored in the respective model variables e.g., $ips \leftarrow TSN_{IPS}$ implies that model variable ips stores the value of source IP address of the TCP segment. Also sequence number of the segment is stored in variable seq_no. Probability of occurrence of τ_1 if system is in state x_0 is $p1$.

- At state x_1, τ_2 implies that another TCP segment (with MSS data bytes) is sent. Here, $check(V) = \{ips = TSN_{IPS}, ipd = TSN_{IPD}, pts = TSN_{PTS}, ptd = TSN_{PTD}\}$. This checking verifies that the current TCP segment belongs to the same TCP connection as that corresponding to τ_1 e.g., $ips = TSN_{IPS}$, ips has stored the value of source IP address of the TCP segment corresponding to τ_1 and here it is checked if ips is same as the source IP of the TCP segment in τ_2. $assign(V) = \{seq\_no \leftarrow TSN_{SN}\}$; model variable seq_no is always updated with the segment number of the latest new (i.e., not retransmit) TCP segment sent with MSS data bytes.

- At state x_1, τ_3 implies that a TCP acknowledgement has arrived from the receiver for some segment that has been sent. $\sigma = TA$ implies that the transition is en-

abled when the SEND_RCV_HANDLER() generates the event TA, implying that an acknowledgement has been received. $check(V) = \{ips = TA_{IPD}, ipd = TA_{IPS}, pts = TA_{PTD}, ptd = TA_{PTS}, TA_{AN} \leq (seq\_no + MSS)\}$. The conditions on the model variables ips, ipd, pts, ptd are used to verify if the acknowledgement segment belongs to the same TCP connection as that of τ_1; e.g., ips (which stores the source IP of the sender) is matched with destination IP address of the acknowledgement (which is destined to the sender). Condition check on model variable seq_no i.e., $TA_{AN} \leq (seq\_no + MSS)$ ensures that acknowledgement number (i.e., TA_{AN}) is not higher than any segment that is not yet sent. The variable seq_no stores the sequence number of the last new segment that is sent. So no acknowledgment should have a number greater that $seq\_no + MSS$; in fact, the highest acknowledgment number can be $seq\_no + MSS$ (corresponding to the last segment sent), till another new segment is sent; so the checking $TA_{AN} \leq (seq\_no + MSS)$.

- At state x_1, τ_5 implies current TCP segment that is to be sent matched RND1 and SEND_RCV_HANDLER() sent the segment after reducing RND2 data bytes. Event generated is TSL. $check(V)$ is same as τ_2. $assign(V) = \{TEST\_FALG \leftarrow 0, seq\_no1 \leftarrow TSN_{SN}\}$. TEST_FALG is made 0 because until attack or normal condition is decided, no segment with data bytes less than MSS is to be sent. Model variable seq_no1 holds the value of the sequence number of the segment that is used to detect the attack (i.e., having some data bytes less). So, model variable seq_no holds the sequence number of the last TCP segment that had MSS data bytes and seq_no1 stores the sequence number of the segment with less data bytes.

- At state x_2, τ_6 corresponds to all acknowledgements that are for segments which are sent before the one with less data bytes. The corresponding component in $check(V)$ is $TA_{AN} \leq (seq\_no + MSS)$.

- At state x_2, τ_7 corresponds to acknowledgement for the segment sent with less (RND2 less than MSS) data bytes. Also, τ_7 corresponds to acknowledgements for other segments (with MSS data bytes) sent following the one with less data bytes. So the acknowledgements will have acknowledgement numbers of the form $seq\_no1 + k \times MSS - RND2$, where k is a positive integer; this is verified by $(TA_{AN} - seq\_no1 + RND2)\%MSS = 0)$ in $check(V)$. $assign(V)$ makes $TEST\_FALG = 1$,

implying that attack detection can start again i.e., SEND_RCV_HANDLER() can again select a random segment and send it with less data bytes.

- At state x_2, τ_9 corresponds to TCP segments which are sent after the segment with less data bytes, until acknowledgement arrives for the segment with less data bytes. It may be noted that in τ_9, $assign(V)$ is $-$ (even sequence number is not stored). After the TCP segment with less data bytes is sent we need not study any more segments that are being sent, rather we need to focus only on the acknowledgement numbers (which is done in transition τ_7). After the acknowledgement corresponding to the segment with less data bytes arrives (in transition τ_7), we restart attack detection and then we again concentrate on sequence numbers of segments being sent.

States x'_0, x'_1, x'_2, x'_3 and transitions $\tau'_0, \tau'_1, \tau'_2, \tau'_3, \tau'_4, \tau'_5, \tau'_6, \tau'_7, \tau'_8, \tau'_9, \tau'_{10}$ represent the system under **attack condition**. We discuss only the transitions which are different in the attack model compared to the normal one.

- At state x'_1, transition τ'_4 corresponds to arrival of an acknowledgement whose number is larger than that corresponding to the last segment sent. This case is possible as the attacker may send acknowledgements before receiving the segments. $check(V)$ ensures this by the equation $TA_{AN} > (seq\_no + MSS)$. It may be noted that seq_no holds the sequence number of the last segment sent whose acknowledgment number is $seq\_no + MSS$ and anything more than this is an attack.

- At state x'_2, transitions τ'_7 and τ'_8 correspond to acknowledgements after the segment sent with less data bytes. Now two cases can occur (i) the attacker may guess that a segment with RND2 less data bytes have been sent and the optimistic acknowledgements are of the form $seq\_no1 + k \times MSS - RND2$, where k is a positive integer. τ'_7 corresponds to this fact; (ii) if attacker cannot make a proper guess then the acknowledgements are of the form $seq\_no1 + k \times MSS$. τ'_8 ensures to this fact by the condition check $(TA_{AN} - seq\_no1 + RND2)\%MSS \neq 0$. Probability of τ'_7 is $p7'$ and τ'_8 is $p8'$. As the attacker needs to guess the segment with less data bytes (i.e., RND1) and also the amount of data bytes less (i.e., RND2), $p8' \approx p7'/(L1 \times L2)$.

- At state x'_3 there is a single transition τ'_{10} which keeps the system in x'_3.

- From state x_0 there is a fault (attack) causing transition τ_{1F_i}.

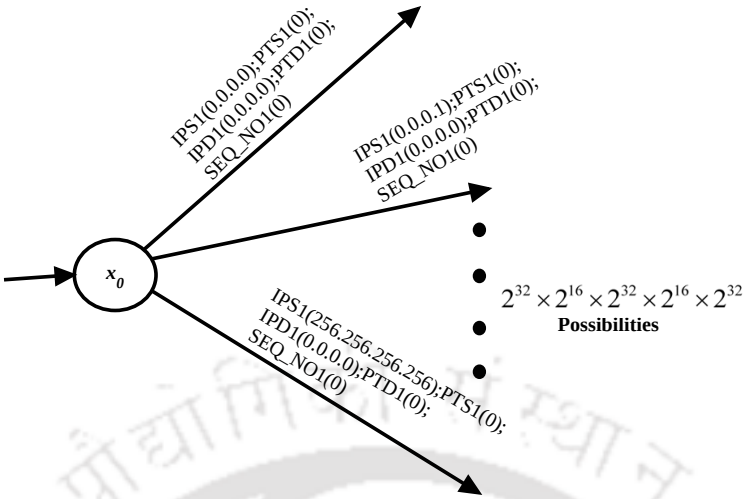

Figure 4.6: Illustration of state explosion problem without model variables

Note: If model variables had not been used, there would have been $2^{32} \times 2^{16} \times 2^{32} \times 2^{16} \times 2^{32}$ transitions (and states) from x_0, each representing a combination of $IPS - PTS - IPD - PTD - seq\_no$ sent in the TCP segment. This would lead to state explosion problem as shown in Figure 5.6.

The model of Figure 4.5 under observability is as follows:

Measurement and Unmeasurable transitions: $\Sigma_m = \{TSN, TSL, TA\}$ and $\Sigma_{um} = \{attack\}$. So, transition τ_{1F_i} is unmeasurable, which implies the fact that attack cannot be detected directly (say by observing a single TCP segment etc.). All other transitions are measurable.

Measurement Equivalent Transitions and States: In Figure 4.5, $\tau_1 E \tau_1'$, $\tau_2 E \tau_2'$, $\tau_3 E \tau_3'$, $\tau_5 E \tau_5'$, $\tau_6 E \tau_6'$, $\tau_7 E \tau_7'$ and $\tau_9 E \tau_9'$. Regarding states, $x_0 E x_0'$, $x_1 E x_1'$ and $x_2 E x_2'$.

Failure Modeling: States and transitions with a prime correspond to failure (or attack), e.g., $C(x_0') = \{F_i\}$ and τ_1' is an F_i-transition. States and transitions without primes correspond to normal (N) condition, $C(x_0) = \{N\}$ and τ_1 is an N-transition.

Diagnosability of the fault: Let us consider a trace $s = \langle \tau_{1F_i}, \tau_1' \rangle$; obviously $s \in \psi(S_{F_i})$. For diagnosing F_i, any sufficiently long but finite extension t say, of s must ascertain that fault has occurred. By Diagnosability definition , if there exist any non-F_i G-trace y say, which is measurement equivalent with st, then probability of st must fall with extension of t. In this case, if we extend $s = \langle \tau_{1F_i}, \tau_1' \rangle$ as $t = \langle (\tau_2', \tau_3')^k \rangle$, where k is arbitrarily large, we get $\exists y = \langle \tau_1, (\tau_2, \tau_3)^k \rangle \in P^{-1}(P(st)) \wedge final(y) = x_1 \notin X_{F_i}$. However, probability of $t = \langle (\tau_2', \tau_3')^k \rangle$ is $(p2' \times p3')^k$ which reduces with increase in k (as $p2', p3' < 1$). s can also be extended as

$t = \langle \tau'_2, (\tau'_5, \tau'_7)^k \rangle$. Here, $\exists y = \langle \tau_1, \tau_2, (\tau_5, \tau_7)^k \rangle \in P^{-1}(P(st)) \wedge final(y) = x_1 \notin X_{F_i}$. However, probability of $t = \langle (\tau'_5, \tau'_7)^k \rangle$ is $(p5' \times p7')^k$ which reduces with increase in k. Similar properties can be shown for all extension traces involving combinations of transitions moving through states x'_1 and x'_2. It may be noted that similar property holds for any trace s in the example. So fault is diagnosable or attack is detectable. In other words, trace involving transitions $\tau'_1, \tau'_2, \tau'_3, \tau'_5, \tau'_6, \tau'_7, \tau'_9$ cannot detect the fault as there exists corresponding transitions in the normal model which are equivalent. Transitions τ'_4 and τ'_8 cause the difference between normal and faulty condition and are mandatory for attack detection. After fault, probability of traces involving states x'_1 and x'_2 decrease and probability of occurrence of τ'_4 and τ'_8 increase, leading to attack detection.

Now we show that the diagnosability condition is satisfied by the stochastic diagnoser for the model shown in Figure 4.5. The stochastic DES diagnoser is shown in Figure 4.7. In the diagnoser, only z_3 is F_i-certain D-state and all others are F_i-uncertain D-states.

Let us consider the D-trace $\gamma = \langle a0, a1, a5, a7 \rangle$. If the diagnoser moves in γ, then either system trace is $\tau_0, \tau_1, \tau_5, \tau_7$ (normal condition) or $\tau_0, \tau_{1F_i}, \tau'_1, \tau'_5, \tau'_7$ (fault condition). The probability of γ is

$$\begin{bmatrix} 1 & \mathrm{p1}_{\mathrm{F_i}} \end{bmatrix} \times \begin{bmatrix} \mathrm{p1} & 0 \\ 0 & \mathrm{p1'} \end{bmatrix} \times \begin{bmatrix} \mathrm{p5} & 0 \\ 0 & \mathrm{p5'} \end{bmatrix} \times \begin{bmatrix} \mathrm{p7} & 0 \\ 0 & \mathrm{p7'} \end{bmatrix} = \begin{bmatrix} 1 \cdot \mathrm{p1} \cdot \mathrm{p5} \cdot \mathrm{p7} & \mathrm{p1}_{\mathrm{F_i}} \cdot \mathrm{p1'} \cdot \mathrm{p5'} \cdot \mathrm{p7'} \end{bmatrix}$$

The D-trace γ leads to z_1, where we have system states x_1, x'_1. The probability of γ is $[1 \cdot p1 \cdot p5 \cdot p7 \quad p1_{F_i} \cdot p1' \cdot p5' \cdot p7']$, which implies that probability of reaching system state x_1 and x'_1 is $1 \cdot p1 \cdot p5 \cdot p7$ and $p1_{F_i} \cdot p1' \cdot p5' \cdot p7'$, respectively. So, if failure has occurred then probability of the diagnoser being in F_i-uncertain D-state z_1 due to γ (corresponding F_i-G-trace $\tau_0, \tau_{1F_i}, \tau'_1, \tau'_5, \tau'_7$) is $p1_{F_i} \cdot p1' \cdot p5' \cdot p7'$. Now if we extend the D-trace γ by D-transitions $a5, a7$ (i.e., by extending the corresponding F_i-G-trace by τ'_5, τ'_7), then probability of the extended D-trace is

$$\begin{bmatrix} 1 & \mathrm{p1}_{\mathrm{F_i}} \end{bmatrix} \times \begin{bmatrix} \mathrm{p1} & 0 \\ 0 & \mathrm{p1'} \end{bmatrix} \times \begin{bmatrix} \mathrm{p5} & 0 \\ 0 & \mathrm{p5'} \end{bmatrix} \times \begin{bmatrix} \mathrm{p7} & 0 \\ 0 & \mathrm{p7'} \end{bmatrix} \times \begin{bmatrix} \mathrm{p5} & 0 \\ 0 & \mathrm{p5'} \end{bmatrix} \times \begin{bmatrix} \mathrm{p7} & 0 \\ 0 & \mathrm{p7'} \end{bmatrix}$$
$$= \begin{bmatrix} 1 \cdot \mathrm{p1} \cdot \mathrm{p5} \cdot \mathrm{p7} \cdot \mathrm{p5} \cdot \mathrm{p7} & \mathrm{p1}_{\mathrm{F_i}} \cdot \mathrm{p1'} \cdot \mathrm{p5'} \cdot \mathrm{p7'} \cdot \mathrm{p5'} \cdot \mathrm{p7'} \end{bmatrix}$$

So, due to extension of γ, probability of the diagnoser being in F_i-uncertain D-state z_1 is

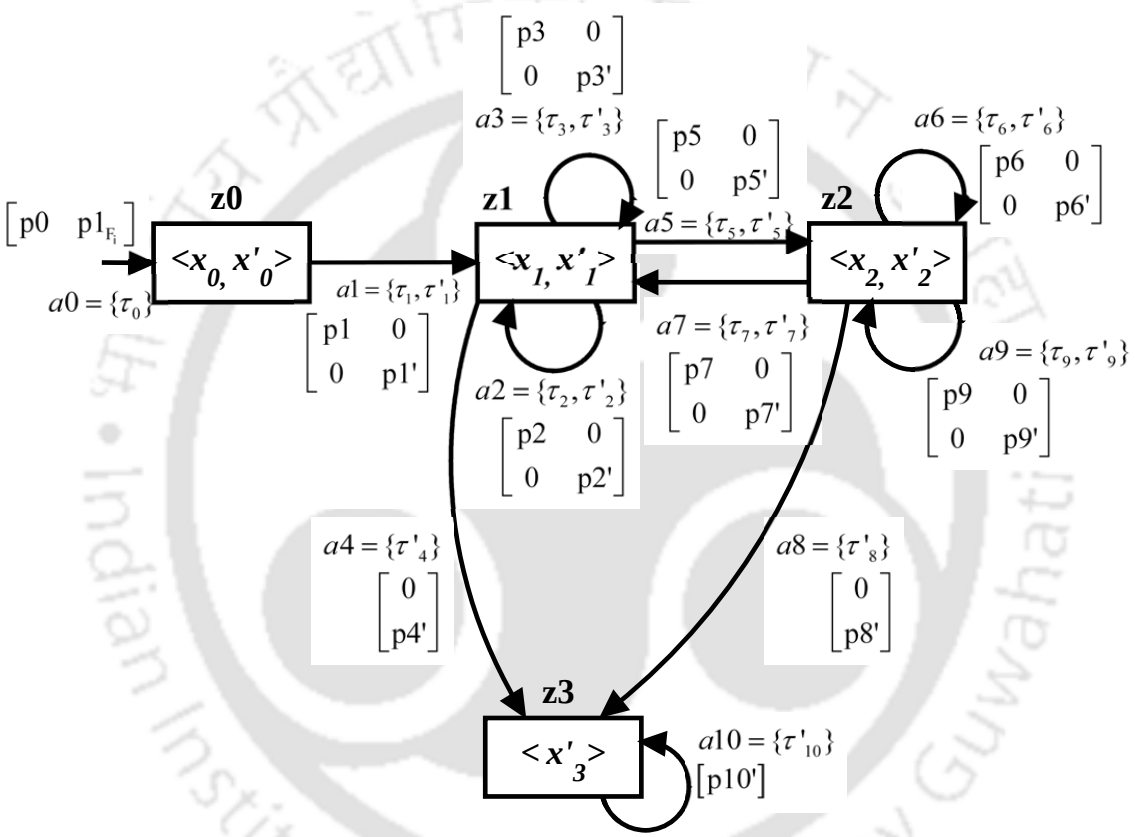

Figure 4.7: Diagnoser for the DES of Figure 4.5

$p1_{F_i} \cdot p1' \cdot p5' \cdot p7' \cdot p5' \cdot p7'$. As $p5', p7' < 1$, so probability of the diagnoser remaining in F_i-uncertain D-state z_1 decreases with its extension in the corresponding F_i-G-trace. Similar can be shown for other D-traces moving in F_i-uncertain D-states.

Now we consider the trace $\gamma 1 = \langle a0, a1, a3, a5, a8 \rangle$, which takes the diagnoser out of F_i-uncertain D-states, leading to z_3, resulting is fault diagnosis. The probability of $\gamma 1$ is

$$\begin{bmatrix} 1 & \mathrm{p1}_{\mathrm{F_i}} \end{bmatrix} \times \begin{bmatrix} \mathrm{p1} & 0 \\ 0 & \mathrm{p1}' \end{bmatrix} \times \begin{bmatrix} \mathrm{p3} & 0 \\ 0 & \mathrm{p3}' \end{bmatrix} \times \begin{bmatrix} \mathrm{p5} & 0 \\ 0 & \mathrm{p5}' \end{bmatrix} \times \begin{bmatrix} 0 \\ \mathrm{p8}' \end{bmatrix}$$
$$= \left[\mathrm{p1}_{\mathrm{F_i}}.\mathrm{p1}' \cdot \mathrm{p3}' \cdot \mathrm{p5}' \cdot \mathrm{p8}' \right]$$

So, probability of reaching x_3' (in z_3) is $p1_{F_i} \times p1' \times p3' \times p5' \times p8'$. Now if $\gamma 1$ is extended as $a10$ (by virtue of corresponding F_i-G-transition τ_{10}'), probability of reaching x_3' (in z_3) is $p1_{F_i} \times p1' \times p3' \times p5' \times p8' \times p10'$. As $p10' = 1$, the probability of the diagnoser remaining in F_i-certain D-state z_3 do not decrease with extension in the corresponding F_i-G-trace; same holds for any other possible extension.

4.4.1 An Example of Attack Detection

In this subsection we illustrate our scheme to detect induced low rate TCP attack using an example. For simplicity, we will not illustrate explicitly the IP address and port numbers. The main concept of the proposed scheme is based on sequence numbers, acknowledgement numbers and DES diagnoser which are highlighted in the example.

Assume that TCP connection has been established and τ_0 has occurred. Let MSS be 100 bytes and the sequence number of the first segment being sent is 1001. In the SEND_RCV_HANDLER(), let L1 be 1000 and L2 be 100. Initially, *TEST_FLAG* = 1. Let RND1 came out to be 2 and RND2 as 30. Then, SEG_NO becomes 1 and as SEG_NO ≠ RND1, the first TCP segment is sent with 1 MSS data bytes. SEND_RCV_HANDLER() generates event *TSN*; transition τ_1 (or τ_1') occurs and *seq_no* becomes 1001. Following that, acknowledgment for the TCP segment arrives (event generated is *TA*) with acknowledgement number as 1101; τ_3 (or τ_3') occurs as $TA_{AN} = seq\_no + MSS$. Now second TCP segment is ready to be sent and SEG_NO becomes 2. As SEG_NO becomes same as RND1 and *TEST_FLAG* is 1, RND2 (i.e., 30) data bytes are reduced in the segment. SEND_RCV_HANDLER() generates event *TSL* and sends the segment with 70 data bytes. τ_5 (or τ_5') occurs and *seq_no*1 becomes 1101. Now let the attacker send an optimistic acknowledgement i.e., send acknowledgement for second segment before receiving it.

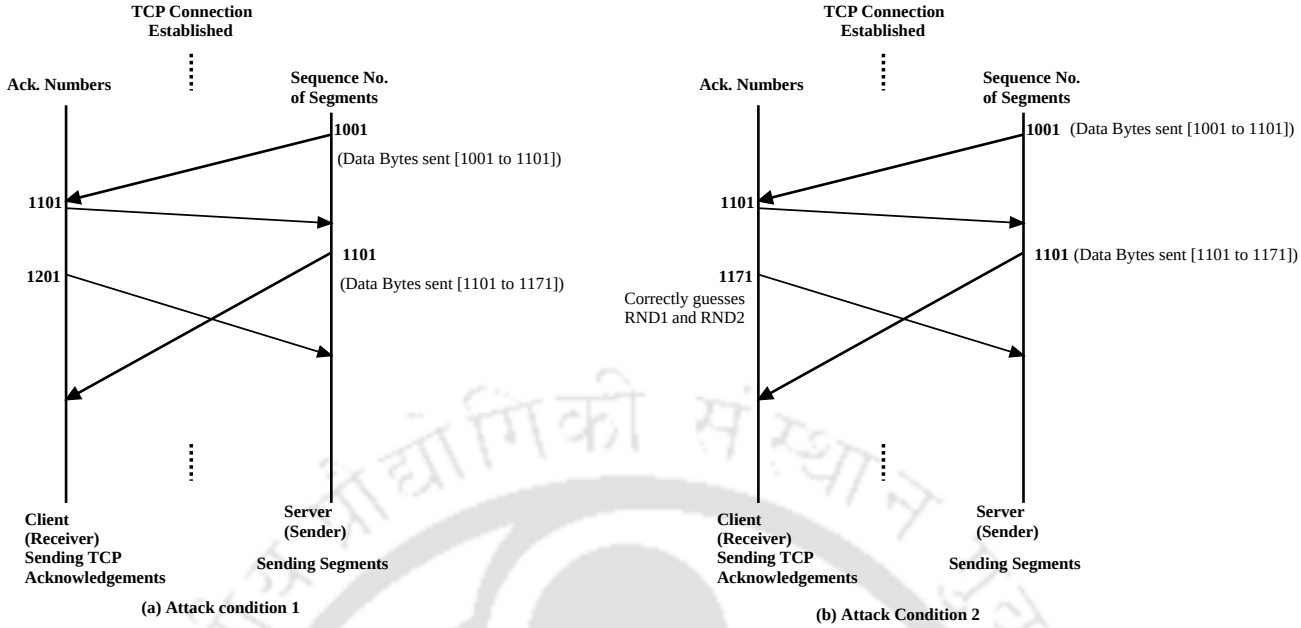

Figure 4.8: Attack detection example scenario

Attacker feels that the next segment will have sequence number as 1101 and MSS data bytes will arrive. So in the optimistic acknowledgement, the acknowledgement number given is $1101 + 1 \times MSS = 1201$. On receiving the acknowledgement, SEND_RCV_HANDLER() generates event TA. Transition τ'_8 occurs because $(1201 - 1101 + 30)\%100 \neq 0$; here $TA_{AN} = 1201, seq\_no1 = 1101, RND2 = 30, MSS = 100$. This situation is shown in Figure 4.8 (a). According to diagnosability definition 4.4, the G-trace corresponding to this situation is $\langle \tau_{1F_i}, \tau'_1, \tau'_3, \tau'_5, \tau'_8 \rangle$. Let $s = \langle \tau_{1F_i}, \tau'_1, \tau'_3 \rangle$ and $t = \langle \tau'_5, \tau'_8 \rangle$. $s \in \psi(S_{F_i})$ and there is no $y \in P^{-1}(P(st)) \wedge final(y) =\notin X_{F_i}$. So, trace st results in diagnosability. In terms of diagnosability condition, the observed D-trace is $\langle a0, a1, a3, a5, a8 \rangle$, which leads to F_i-certain D-state z_3 where fault is detected. According to attack detection, as the sequence number in the acknowledgement is of the from "s_no + multiple of MSS", even for the segment with less data bytes, attack is detected.

Now let us assume that attacker correctly guesses RND1 and RND2 and sends acknowledgement with acknowledgement number as 1171. On receiving the acknowledgement, SEND_RCV_HANDLER() generates event TA. Transition τ_7 (or τ'_7) occurs because $(1171 - 1101 + 30)\%100 = 0$; here $TA_{AN} = 1701, seq\_no1 = 1101, RND2 = 30, MSS = 100$. This situation is shown in Figure 4.8 (b). τ_7 (or τ'_7) makes $TEST\_FLAG$ 1 and the whole attack detection process starts again. It may be noted that under this condition we cannot

ensure whether attacker is guessing correctly or the acknowledgement is not optimistic. So attack or normal condition cannot be determined with certainty. This situation is shown in Figure 4.8(b).

Let us analyze this situation according to diagnosability definition. Here, the G-trace is $\langle \tau_{1F_i}, \tau'_1, \tau'_3, \tau'_5, \tau'_7 \rangle$. Let $s = \langle \tau_{1F_i}, \tau'_1, \tau'_3 \rangle$ and $t = \langle \tau'_5, \tau'_7 \rangle$. $s \in \psi(S_{F_i})$ and $\exists y = \langle \tau_1, \tau_2, \tau_5, \tau_7 \rangle \in P^{-1}(P(st)) \wedge final(y) = x_1 \notin X_{F_i}$. So, trace st is responsible for non-diagnosability, which maps to the fact that attacker has correctly guessed RND1, RND2 and sent a proper acknowledgement. So attack could not be detected. Now let us assume that in the next attack detection cycle, the attacker again guesses RND1, RND2 properly. The G-trace now is $\langle \tau_{1F_i}, \tau'_1, \tau'_3, \tau'_5, \tau'_7, \tau'_5, \tau'_7 \rangle$. Let $s = \langle \tau_{1F_i}, \tau'_1, \tau'_3 \rangle$ and $t1 = \langle \tau'_5, \tau'_7, \tau'_5, \tau'_7 \rangle$. In this case also $\exists y = \langle \tau_1, \tau_2, \tau_5, \tau_7, \tau_5, \tau_7 \rangle \in P^{-1}(P(st1)) \wedge final(y) = x_1 \notin X_{F_i}$. So, trace $st1$ is also responsible for non-diagnosability. However it may be noted that $prob(st1) < prob(st)$ (as because $p5', p7' < 1$), implies that probability of traces which cause non-diagnosability reduce with increase in length, satisfying diagnosability definition. Now this situation is examined according to diagnosability condition. D-trace for this situation is $\langle a0, a1, a3, a5, a7 \rangle$ leading to F_i-uncertain D-state z_1; so fault is non-diagnosable. Corresponding to this D-trace the G-traces are $\langle \tau_1, \tau_3, \tau_5, \tau_7 \rangle$ (normal condition) or $\tau'_1, \tau'_3, \tau'_5, \tau'_7$ (fault condition). The post G-trace $t = \tau'_5, \tau'_7$ is responsible for keeping the diagnoser in F_i-uncertain D-state z_1 (under fault) and its probability falls with increase in length of t. So the probability of remaining in F_i-uncertain D-state z_1 falls with increase in length of the F_i-G-trace $\tau'_5, \tau'_7, \cdots \tau'_5, \tau'_7, \cdots$; so diagnosability condition is not violated. In the attack detection context, the probability of guessing RND1 and RND2 correctly in all attack detection rounds fall with increase in the number of rounds. So it can be stated that "probability that an attacker will fail to guess RND1 and RND2 at some attack detection cycle is fairly high", leading to its detection.

4.5 Experimentation and Results

The metrics for efficiency of an attack detection/mitigation system are accuracy and detection rate. Also, the impact of attack in presence and absence of the detection/mitigation system is important. In this section we report accuracy and detection rate of the proposed scheme for different ranges of the two random numbers (i.e., L1 and L2 in SEND_RCV_HANDLER()). Also, impact of the attack (in terms of loss in throughput of a genuine TCP connection to

the server, which is under attack), with and without the detection mechanism, is presented. A test bed has been created for verifying the proposed scheme; the topology is shown in Figure 4.9. The testbed consists of switches, 4 routers, a server, an attacker and a genuine host. The operating system in the server is Fedora 14, genuine host is Ubuntu 11 and attacker is BackTrack 4.

We have already discussed enabling (based on events and conditions on model variables) and firing of model transitions (Figure 4.5 and Table 4.1). Also we have discussed assignment of model variables in the transitions at the model level. However, finally the DES diagnoser (not the model) is implemented as a software module and used as attack detector, as shown in Figure 4.4. Enabling of the D-transitions and assignment of model variables in D-transitions are same as any one of the model transitions contained in the corresponding D-transition. It may be noted that a D-transition contains equivalent model transitions. So all model transitions in a D-transition have same event, same condition on model variables and same assignment of model variables. In other words, a D-transition works similar to any one of the model transition it comprises.

All the attack certain D-states give an output of 1, and other states output 0. For each TCP connection established, an instance of the diagnoser at state z_0 is spawned and is added to an active list. On the arrival of next event TSN, TSL, TA, it is given to all instances of diagnosers in the active list to their corresponding present state. Each of them can make a transition on these events if the enabling condition is satisfied. Any diagnoser that reaches an F_i certain D-state or probability of attack becomes higher than the threshold, are deleted from the list.

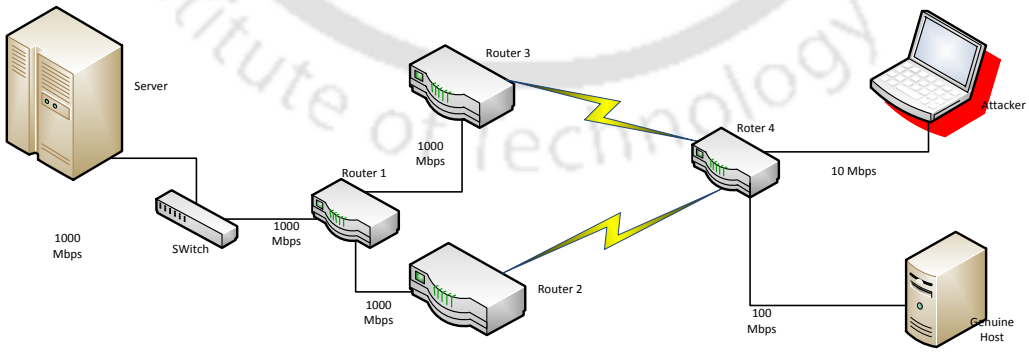

Figure 4.9: Test Bed setup for the experiments

First we present the throughput (in KB) of the TCP connection between the genuine

host and server when:

1. Without attack (i.e., attacker not sending optimistic acknowledgements to server). Data transfer through TCP was performed between genuine host and server. Throughput of the connection at regular time interval of 0.2 seconds is examined and total throughput at each second is reported in Figure 4.10 (solid line).

2. With attack (i.e., attacker sending optimistic acknowledgements to server). First, a TCP connection is established between genuine host and server. Another TCP connection is established between attacker and server, and it sends optimistic acknowledgements. The throughput of the connection between genuine host and server is shown in Figure 4.10 (dotted line). Obviously, due to induced low rate TCP attack there is substantial fall in throughput.

3. With attack and the proposed detection engine in execution in the server. This case is same as (2), but the proposed attack detection engine is executed in the server when the TCP connections are established. On detection of attack by DES diagnoser, the connection between server and attacker is dropped. The throughput of the connection between genuine host and server is shown in Figure 4.11 (dotted line). For comparison with normal throughput, in the same figure we also show the throughput without attack condition using the solid line. It may be noted that there is marginal difference in throughput.

From the graphs in figure 4.10,4.11 it may be observed that the induced low rate attack could reduce the throughput considerably. So the attack may lead to DoS. However, after the proposed solution is applied, the impact of attack on throughput is minimal. Also, it may be noted that the proposed detection mechanism has a small performance penalty in terms of loss of throughput because few data bytes are reduced in random segments. This performance penalty is also taken into account in the throughput calculation shown in Figure 4.11 (dotted line). So it may be concluded that overhead of the proposed system is minimal.

Note: The accuracy is always 100% (i.e., false positives are 0) because attack is detected when an acknowledgement arrives whose (i) acknowledgment number is larger than "sequence number of last sent segment + MSS"; (ii) acknowledgment number is "sequence

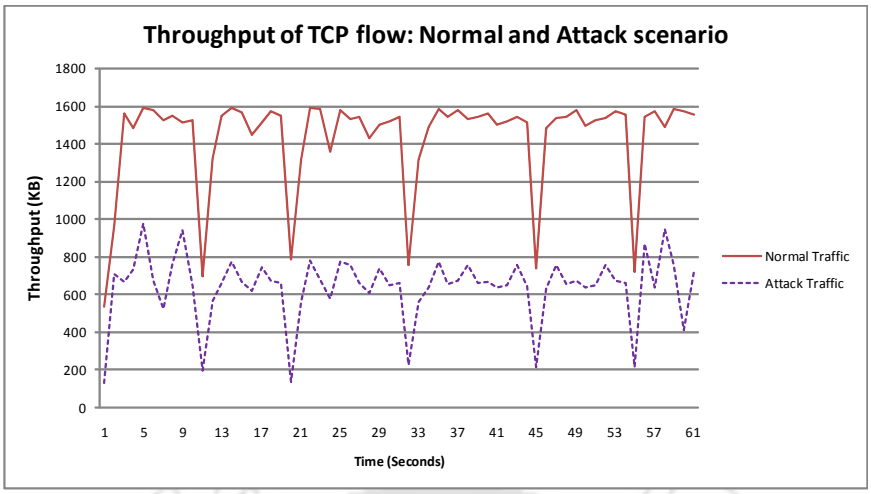

Figure 4.10: Throughput of TCP flow: Normal and Attack scenario

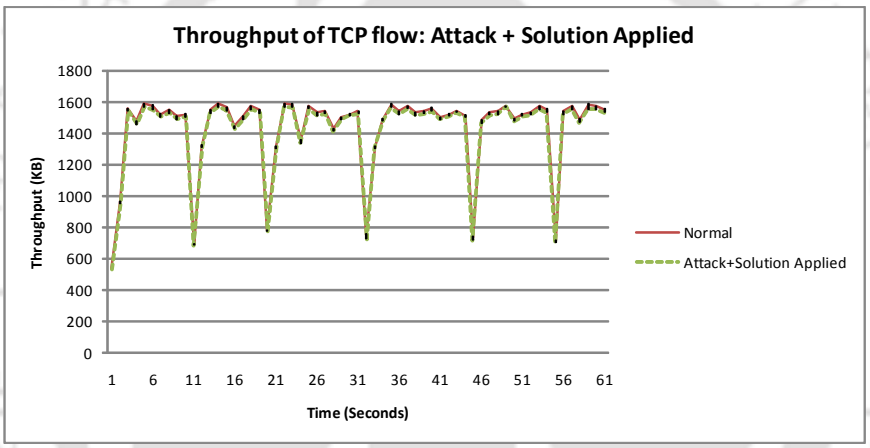

Figure 4.11: Throughput of TCP flow: Attack + Solution Applied

number of last sent segment + $k \times MSS$" (k is a positive number), but a segment has been sent with some data bytes less than MSS. It may be noted that these two cases do not happen in normal conditions.

Tables 4.2, 4.3 and 4.4 show detection rate and throughput for different ranges of the random numbers (L1 and L2). Table 4.2 reports detection rate and throughput for three ranges of L1 namely, 1-100, 1-300, and 1-1000 when L2 is between 1-50.

From the table 4.2, 4.3 and 4.4 it may be observed that larger the range of $L1$, lower is the detection rate. This is because attempt of attack detection (i.e, selection of one segment per L1 segments with less data bytes) occurs less frequently when L1 is large. In some of the cases the attacker may guess the random numbers leading to false negatives (i.e., lower detection rate). However with increase in L1, throughput of the TCP connection is higher;

Table 4.2: Detection rate and Throughput for different ranges of L1 when L2 is between 1-100

L1	Throughput Normal	Throughput Attack + Solution	Detection rate
1-100	1569.66 KB/sec	1541.21 KB/sec	100
1-300	1569.66 KB/sec	1552.35 KB/sec	99.57
1-1000	1569.66 KB/sec	1559.72 KB/sec	95.11

for comparison, in the table we also present the throughput under normal condition (and without the proposed solution in execution). This is because, higher the value of L1, less is the number of segments sent with less data bytes. However, it may also be noted that drop in throughput is not substantial for range of L1 between 1-100 compared to the range 1-1000.

Table 4.3: Detection rate and Throughput for different ranges of L1 when L2 is between 1-300

L1	Throughput Normal	Throughput Attack + Solution	Detection rate
1-100	1569.66 KB/sec	1536.59 KB/sec	100
1-300	1569.66 KB/sec	1549.59 KB/sec	99.81
1-1000	1569.66 KB/sec	1555.59 KB/sec	95.15

Table 4.4: Detection rate and Throughput for different ranges of L1 when L2 is between 1-1000

L1	Throughput Normal	Throughput Attack + Solution	Detection rate
1-100	1569.66 KB/sec	1529.62 KB/sec	100
1-300	1569.66 KB/sec	1545.38 KB/sec	99.87
1-1000	1569.66 KB/sec	1553.59 KB/sec	95.21

Table 4.3 and Table 4.4 report detection rate and throughput for the same three ranges of L1, when L2 is between 1-300 and 1-1000, respectively. From the tables, it may be noted that higher the range of L2, higher is the detection rate. This is because higher the value of L2, more difficult it is for the attacker to guess the number of data bytes less in the segment. However, higher the range of L2, lower is the throughput. This is because L2 impacts the number of bytes to be reduced in the segments. However, reported results illustrate that this impact is not substantial; drop in throughput is about 3% in the worst case, when L1 is 1-100 and L2 is 1-1000.

As our experiments illustrate that impact of low L1 and high L2 is minimal on TCP

throughput but achieves 100% detection rate, we may conclude that the proposed scheme can provide very high detection rate at minimal overheads.

4.6 Conclusion

In this chapter, Stochastic DES framework [36] has been adapted and used for building an attack detection scheme for induced low rate TCP attack. In this attack, the attacker first establishes a TCP connection with a server and sends acknowledgements to the server even for segments it has not yet received; this is called optimistic acknowledgements. The server believes that the route to the attacker is free of congestion and increases the rate of transfer leading to congestion and DoS to other genuine TCP connections. It is discussed that as the attack is relatively new only a few solutions have been proposed for its detection/mitigation. However, the main issues with these schemes include lack of scalability, change in TCP header format, lack of formal frameworks for design of the attack detector etc. Following that, it was highlighted, how the proposed stochastic DES based attack detection scheme can address these issues. It is also shown that the stochastic DES framework [36] needs to be augmented with model variables to make it capable of modeling and detecting the induced low rate TCP attack. Finally, the design procedure of the proposed attack detector was explained and illustrated using an example. The scheme is verified in a test bed and results illustrated high detection rate and 100% accuracy with little compromise in TCP throughput.

The attacks discussed till now have been efficiently detected using different DES frameworks namely, Active, I-Diagnosis and Stochastic. However, their effectiveness depend on the correctness of the DES models. It may be noted that in all these DES frameworks, the modeling step is manual which involves building DES models from normal (protocol) and attack specifications, which are in natural language. So if the normal protocol or the attack is complex, the effectiveness of the IDS may be compromised due to chances of errors in the DES models. This motivated us to explore Linear Time Temporal Logic (LTL) based DES for detecting NDP related attacks. NDP and related attacks are complex (compared to the ones dealt with, till now), however, LTL based DES framework facilitates easy translation of natural language specifications to LTL formulas and checking correctness of each step involved in detector construction.

Chapter 5

LTL in DES based detection of NDP related attacks

5.1 Introduction

In a network, hosts or computers are identified by their Internet Protocol(IP) addresses. Precisely IP addresses are used by the network layer for identifying the machine uniquely. With the unprecedented growth in the number of connected hosts in the Internet the demand for IP address has increased manyfold. The 4.3 billion addresses provided by IPv4 is facing near exhaustion. To meet this demand, organizations are migrating to IPv6 address which provides much larger address space of 340 undecillion or 3.4 x 10^{38} addresses. This expansion allows for many more devices and users on the Internet as well as extra flexibility in allocating addresses and efficiency for routing traffic [65] etc.

IPv6 uses two addressing schemes, one for local communication on the link (called *link local address*) and the other for communication outside the link (called *global address*). IPv6 subnet size has been standardized by fixing the size of the host identifier to 64 bits to facilitate an automatic mechanism for generating the host identifier from MAC address. The link local address is 128 bits and is generated as follows. The lower order 64 bits of link local address is obtained by inserting a fixed value of 0xFFFE (comprising 16 bits) in the middle (i.e., 25^{th} to 41^{th} position) of the 48 bit MAC address of the host and changing the 7^{th} bit of MAC from 0 to 1; these 64 bits are called *interface identifier*. The higher order 64 bits of the link local address is a fixed value of 0xFF80:0000:0000:0000. The link local

IPv6 address identifies a host in the link and can be used to communicate with other hosts on the link. The global IPv6 address is used to uniquely identify a host in the Internet. The lower order 64 bits of the global address for a host is same as the lower order 64 bits of the link local address. The higher order 64 bits are obtained from an incoming Router Advertisement packet.

In IPv6, Network Discovery Protocol (NDP) is used for finding the MAC address given the IP address. NDP is a stateless protocol and lacks authentication of its messages by default. Lack of authenticity and statelessness resulted in possibility of many attacks. In IPv4, as ARP provides only the basic translation of IP to MAC address, there were only two kinds of spoofing possible, request and response spoofing. In IPv6, as NDP provides more features than just translation of IP to MAC, several kinds of spoofing are possible. The prevalent attacks based on NDP are Neighbor Solicitation(NS) spoofing, Neighbor Advertisement(NA) spoofing, Duplicate Address Detection(DAD) spoofing, Neighbor Unreachability Detection(NUD) spoofing etc. Many attacks in the network like Man-in-the-Middle (MiTM) and Denial of Service(DoS) can be launched using the attacks mentioned above.

Unlike IPv4 network, there are not many options available to prevent or detect NDP based attacks as IPv6 protocol is relatively new and slowly coming in use. Recently, few techniques have been proposed for detection/prevention of these attacks. However, these techniques have several drawbacks namely, non-scalability, computationally expensive, management of cryptographic keys, change in the NDP itself etc.

In this chapter we propose to adapt the Linear Temporal Logic (LTL) based DES framework to design an IDS to combat against NDP attacks. Applying the LTL based DES framework for IDS, first requires modeling of the network under normal and attack(s) condition. Following that, the normal condition of the protocol is expressed as an LTL formula. Then the detector is designed using system model and the LTL formula. The DES detector acts as the IDS, which observes sequence of NDP based packets and detects attack if the sequence violates the LTL formula.

NDP related attacks are similar to ARP based attacks. So active DES framework [21] used for ARP based attacks (in Chapter 2) could have been applied for NDP related cases also. It may be noted that in all the DES frameworks discussed in the last three chapters, modeling of the normal and attack conditions are done manually, which is assumed

to be correct. The effectiveness of the corresponding IDSs are dependent on the DES modeling. Specification of NDP and the related attacks are more complex compared to their counterpart in ARP. So if active DES framework is applied to design the IDS for NDP attacks, there are chances of compromise in effectiveness due to the manual modeling step involved.

To address the issue of manual modeling in state based DES frameworks [20, 21], Jiang et al. in [43] proposed LTL based DES paradigm. As in case of state based models, in the LTL-based DES framework also, first the system under normal and attack conditions are developed manually. Following that, the normal system specification is expressed as LTL formulas. The LTL formulas are then converted into a Finite State Automata (FSA) called Buchi Automata using automated tools [44]. Although, modeling and specification conversion to LTL formulas are done manually, automated methods like model checking are used to verify the LTL specifications vis-a-vis the system model [66]. Correctness of the LTL formulas and the system model can be determined before progressing to detector design. The detector in LTL based DES paradigm is obtained by proposition synchronization of Buchi automata and the system model. The detector observes system traces and detects a fault if the trace violates the LTL formula.

Several extensions are required in the LTL based DES framework [43] to make it applicable in IDS for NDP attacks. The extensions and the corresponding motivations are as follows.

- *Handling state space explosion:* In the LTL based DES framework [43], LTL specifications are modeled (automatically) in terms of Buchi Automata which is a state transition system. Also, the system and the detector are modeled as a state-transition systems. Broadly, speaking most of the steps in the LTL based DES framework [43] are in form of state-transition systems termed as Finite State Automata (FSAs). The major issue with FSAs is that they suffer from the state explosion problem if the system to be modeled is complex. NDP is responsible for mapping IP addresses to MAC addresses, so IP and MAC addresses are the most fundamental parameters that need to be modeled. As the ranges of IP addresses and MAC addresses are very high and theoretically any IP address can be associated with any MAC address (by the attacker), the FSA model may have extreme large number of states. So the LTL based framework cannot be directly applied for modeling NDP and related attacks.

In this chapter we augment the LTL framework with *model variables*, which preserves the advantages and at the same time addresses the issue of state explosion. The model variables are associated with each transition, where they can be assigned values and checked for equality for firing of the corresponding transition. In case of modeling the NDP, we have model variables for IP address and MAC address.

- *Use of active probing:* In FDD using the DES framework, inherently the system under failure has some transition sequence which is different compared to the normal condition. However, in case of NDP based attacks, there is no difference in the sequence of NDP events compared to normal situations. To handle this situation, an active probing mechanism is used so that sequences of packets are different under spoofing and normal conditions. It may be noted that the probing technique maintains the standard NDP.

The proposed IDS for NDP based attacks based on LTL based DES framework has the following salient features.

- LTL framework provides a user-friendly paradigm for transforming natural language specification (normal/ attack condition) of NDP protocol to formal models and formal (LTL) specifications. Thus modeling becomes formal and less error-prone.
- Automated techniques like model checking can be employed to verify correctness of each intermediate step of developing the IDS, ranging from system modeling to detector design. Further, it can be ascertained, that all possible attack scenarios can be detected by the IDS. It only requires to be checked whether for a packet sequence (under attack scenario), there exists no path in the detector.
- Polynomial number of states in the DES detector (which is the IDS) with respect to number of model states. So the IDS can be implemented efficiently in terms of execution time and memory.
- Involves installation of the DES detector (based network IDS) in just one host in the network.
- Follows standard NDP and does not violate the principles of network layering structure.

- Detects a large set of NDP attacks namely, NS spoofing, NA spoofing, DAD attack etc.

The rest of the chapter is organized as follows: In Section 5.2 NDP is first discussed in brief. Following that NDP based attacks and existing schemes for their detections are discussed. The motivation for using the LTL based DES framework for detecting NDP attacks is also pointed out in the same section. Section 5.3 describes the LTL based DES framework [43], when augmented with model variables. It starts with a discussion on preliminaries of LTL. The notions of diagnosability and pre-diagnosability are explained and the steps for building the detector are enumerated. The method for proving correct specification is also stated. Subsequently, in Section 5.4 the framework is applied to model NDP attacks and its detection is exhibited. The same section also discusses pre-diagnosability checking and diagnosability checking of the system in an automated way using the NUSMV Model checking tool. Experimental results are discussed in Section 5.5. The chapter is finally concluded in Section 5.6.

5.2 NDP Related Attacks and Existing Counter Measures

5.2.1 NDP Related Attacks

5.2.1.1 Neighbor Solicitation/Advertisement Spoofing

When a source node wants to communicate with a destination node in the same link, the source node needs to know the MAC address of the destination node. Nodes on the link use Neighbor Solicitation and Advertisement messages to create bindings between IP addresses and MAC addresses. Each host has a neighbor cache which keeps entries for IP-MAC pairings. So a source node first looks in its neighbor cache to check if the MAC address corresponding to the IP address under question (of the destination node) is available. If the MAC address is not in the neighbor cache it sends an NS packet to the multicast address asking the MAC address of the host having the IP address under question. All hosts receive the NS packet (as it is multicast) and the host having the IP address under question sends an unicast NA packet mentioning its MAC address. On receipt of the NA packet, the source node updates its neighbor cache with the MAC address

without any verification. Further, (all) nodes which receive the NS packet, update their cache with IP-MAC pairing information (of the source node) available in the NS packet. This is basically a performance optimization because the nodes can know IP-MAC pairings without explicitly sending an NS packet and receiving an NA packet.

As IP-MAC pairing information from NS and NA packets are accepted without any verification, attackers can easily spoof NS and NA packet with falsified IP-MAC pairings. NS/NA spoofing involves a malicious node sending NS/NA messages to a target node having falsified IP-MAC pairings. Since NDP is a stateless protocol and the cache always updates its entries, the target host blindly writes its neighbor cache with the spoofed IP-MAC pairing. This results in redirecting all the data link layer frames to the spoofed MAC address. For example, let there be three hosts in a link A,B and D having IP address as IP(A),IP(B) and IP(D), respectively; let the MAC address be MAC(A), MAC(B) and MAC(D), respectively. Let A,B be genuine hosts and D be the attacker. Also, let A sent an NS packet to query about MAC address corresponding to IP(B). In response, B will send an NA packet which contains MAC(B). Following this genuine NA packet, attacker D can send a spoofed NA packet to A having IP(B)-MAC(D) (i.e., IP of B associated with its MAC). Neighbor cache of A will update IP(B)-MAC(B) with (falsified) IP(B)-MAC(D). Now all traffic A wants to send B will go to D (as commutation is by MAC address). Similar situations can also be achieved by attacker sending unsolicited NA or NS packets having falsified IP-MAC pairs. This spoofing mechanism can be used for a DoS attack by specifying an unused MAC address. Also this can be used to create Man in the middle (MiTM) attack by spoofing a pair of host's MAC address with the attacker's MAC address.

5.2.1.2 Duplicate Address Detection (DAD) attack

In IPv6, when a new node wants to come up in the network, before using the IP address, it verifies if there is another node which is using the same IP address or it is contending for the same address; this is called duplicate address detection (DAD) attempt. If no such node is found, the new node can use the IP address. Otherwise, another alternative IP address is tried or static IP address is configured. The concept that reply of a node stating "it is using the same IP address or it is contending for the same address" is not verified, can be used by an attacker to launch DoS attack. If the attacker responds to every DAD attempt made by an entering host, then the host will never be able to obtain an address.

The attacker can claim the address in two ways:
(1) Attacker can either reply with an NS packet, simulating that it is also performing DAD for same address. For example, node A say, is trying to configure itself with $IP(A)$ and so it sends a DAD request with target IP address as $IP(A)$. Now an attacker D can also send a DAD request for same IP address ($IP(A)$). On reception of such a DAD request, host A fails to get the target IP address and backs off. This threat was identified in RFC 4862 [37] which specifies that both machine preforming DAD for same address should back off. If attacker D sends a DAD request for all IP addresses A want to configure, there will be a DoS attack on A.
(2) Attacker can reply with an NA packet, simulating that it has already taken the address into use. For example, when node A is trying to configure itself with $IP(A)$ then attacker D can also sends a NA packet for same IP address saying that it has already configured $IP(A)$. On reception of such an NA packet, A backs off thinking that $IP(A)$ is already taken. Similar to case of (1), there will be a DoS attack on A.

5.2.2 Existing Detection Mechanisms

As IPv6 is new, only a few techniques to detect NS/NA spoofing has been reported in the literature. In this sub-section we discuss these techniques and point out issues therein. Following that we present the motivation of the proposed approach.

Use of IPSec [67]: IPsec Authentication Header (AH) can be used with NDP (NS/NA) messages to enhance security and verify through AH that messages do contain proper and accurate information. Security Associations (SA)s can be created using the Internet Key Exchange (IKE). But IKE requires a functional IP stack in order to function and this result in a bootstrapping problem. So SA can only be configured manually, which is a tedious or impractical task considering the volume. Even if SAs can be established, it is not possible to verify the ownership of dynamically generated IP addresses.

Secure Neighbor Discovery (SEND): RFC 3971 [40] defines this mechanism, which uses (PKI) to sign NS/NA messages. It may be noted that key management in a LAN is cumbersome and difficult even for a medium scale organization to implement it.

Cryptographically generated addresses (CGA) [41]: CGA are used to avoid spoofing on the local network. However, this protocol is not yet widely implemented and the

overhead associated with it can cause DoS conditions itself. Further, the scheme requires modification of NDP.

Neighbor Discovery Protocol Monitor (NDPmon) [42]: It is a tool that observes NS/NA packets in the local network to see if there are changes in IP-MAC pairings; on detection of changes it notifies the administrator by writing in the syslog. The problem with this approach is, if the first sent packet itself is having a spoofed MAC address then the whole system fails. Further, any genuine change in IP-MAC pair is discarded.

Windows SEcure Neighbor Discovery (WinSEND)[68]: Rafiee et. al. in [68] implemented SEND as a service for Windows operating system. It is a user-space implementation which is developed in Microsoft .NET. It is claimed to the first SEND implementation for Windows. WinSEND suffers from the same limitations as of SEND. Besides, this being is a platform specific implementation, it requires patching of individual hosts. The purpose gets defeated if the patch is not applied in some host. Therefore WinSEND may not be suitable as it is only for Windows operating system and requires patching of all hosts in the network.

All the techniques discussed above to detect/mitigate NDP attacks are passive techniques. Broadly speaking, the major issues with these techniques comprise non-dynamism, performance penalty of cryptographic computations, patching each host in the LAN etc. To address these issues, we propose an active detection mechanism similar to ARP. The basic idea used in the active IDS for NDP attacks involve sending probe packets to hosts in the LAN in addition to observations (like changes of IP-MAC pairs). It may be noted that development of these active IDSs require a number of manual procedures namely, modeling the attacks from English langue specifications of the protocols and vulnerabilities, IDS design etc. Although the protocol and vulnerabilities are complex but these IDSs lack formal modeling and verification paradigm.

So, from the above issues it may be stated that an efficient IDS for detecting NDP attacks should be verifiable for correctness, dynamic, scalable, covering a large set of attacks etc. Most of these parameters are achievable through LTL based DES framework.

5.3 LTL Based FDD in DES

In this section we present the LTL based DES framework of [43] augmenting with *model variables*, which makes it applicable for design of the IDS for NDP attacks.

5.3.1 Introduction to LTL

Linear-time Temporal Logic (LTL) is a temporal logic based framework where time is treated as linear and modeled as a sequence of states extending infinitely into future. It provides connectives that allow to refer to some future state in the timeline. Temporal logic provides an expressive and natural language for specifying the behavior of any system. In other words, it helps to convert informal natural language specification to an LTL formula in an intuitive fashion. The syntax and semantics of LTL are briefly discussed below.

In Backus Normal Form LTL has the following syntax [69]:
$\phi ::= True|False|p|(\neg\phi)|(\phi \wedge \phi)|(\phi \vee \phi)|(\phi \rightarrow \phi)|(X\phi)|$
$(\mathbb{F}\phi)|(G\phi)|(\phi \cup \phi)|(\phi W\phi)|(\phi R\phi)$ where,
p = atomic proposition from set of atoms.
X = represents next state.
$\mathbb{F}$ = some future state.
G = globally for all states.
$\cup$ = until some future state.
R = release at some future state.
W = weak until.
All these connectives are not required to form an LTL specification as there exists some redundancies among the connectives. A subset of the connectives {True, ¬, ∧, X, ∪} constitute an adequate set. The precise meanings of the connectives forming the adequate set are as follows:

- True, ¬ and ∧ have the same meaning as in propositional logic where ¬*statement* means negation of the *statement* is true and *statement*1 ∧ *statement*2 means that both statement1 and statement2 are true in the current state of time.

- Xp means proposition p is true in next state of time.

- $p \cup q$ means, p is true continuously until q becomes true. q is guaranteed to become true at some future state of time.

The normal behavior of the system is represented by an LTL formula. As per the syntax of temporal logic, an LTL formula can be generated by following rules:

1. an atomic proposition is an LTL formula.
2. if f is an LTL formula, then $\neg f$ is also an LTL formula.
3. if f and f' are LTL formulae, then $f \vee f'$ is also an LTL formula.
4. if f is an LTL formula, then Xf is also an LTL formula.
5. if f and f' are LTL formulae, then $f \cup f'$ is also an LTL formula.

LTL based DES framework translates the LTL specification to a Buchi automaton. The construction process for building the Buchi automata can be found in [70]. Following that several operations are performed on the Buchi automaton and the state-transition model of the system to generate the detector. So, this framework involves state-transition systems defined as:

$$G = (S, R, AP, L) \text{ where,}$$

- S is the finite set of states.
- $R \subseteq S \times S$ is a total transition relation that means $\forall s \in S, \exists s' \in S$ such that $(s, s') \in R$.
- AP is a finite set atomic propositions. Each proposition can take any one of the boolean values between True or False.
- L is a labeling function and this function is defined as $L : S \rightarrow AP$. The label at each state represents the propositions that are true at that state.

Definition 5.1 (State Trace): *A finite or infinite sequence of states such as* $\{s_0, s_1, ...\}$ *is called a state trace or simply a path if for every* $i \in \{0, 1, ...\}$, $(s_i, s_{i+1}) \in R$.
If $t = s_0, s_1, s_2, ...$ *represents a path, then* t^i *denotes a suffix of it starting at state* s_i. *For example,* $t^2 = s_2, s_3, s_4, ...$

Definition 5.2 (Proposition Trace): *A finite or infinite sequence of atomic propositions* $\{p_0, p_1, ...\}$ *where* $p_i \subseteq AP$, $i = 0, 1, ...$ *is called a Proposition Trace. A proposition trace is said to be contained in G if there is a state trace* $s_0, s_1, ...$ *associated with this proposition trace such that* $p_i = L(s_i)\ \forall i \geq 0$.

Using the syntax of LTL and the above definitions, now the semantics of LTL is discussed.

Let $t = s_0, s_1, ...$ be a path in G and f be an LTL formula. $t \models f$ denotes that the *path t satisfies f*. Satisfiability is defined inductively as follows:

- $t \models True$
- If p is an atomic proposition then $t \models f$ iff $p \in L(s_0)$
- $t \models \neg f$ iff $t \not\models f$
- $t \models f_1 \vee f_2$ iff $t \models f_1$ or $t \models f_2$
- $t \models Xf$ iff $t^1 \models f$, where the path $t^1 = s_1, s_2, ...$
- $t \models f_1 \cup f_2$ if there exists $i \geq 1$, such that $t^i \models f_2$ and for all $j = 1, ..., i-1$, $t^j \models f_1$ holds

Above statements define satisfaction relations between paths and LTL formulas. However, the following definition throws light on satisfaction relation between a system and an LTL formula.

Definition 5.3 (Satisfiability): *Given a state* $s \in S$ *in transition model G, an LTL formula* f *is said to be satisfied at s if for every path t starting at s,* $t \models f$. *Now* f *is said to be satisfied at G if for every state* $s_0 \in S_0$, *where* $S_0 \subseteq S$ *is the set of initial states of G,* f *is satisfied at* s_0.

Given the system model and its failure conditions, an LTL specification can be written to represent the non-faulty behavior of the system. Any infinite state trace of the system is called faulty if it violates the LTL specification. In the failure diagnosis mechanism, a detector is built that raises an alarm if a faulty state trace is executed.

5.3.2 Construction of the DES Detector in LTL framework

In this subsection we present the algorithm to construct the DES diagnoser for the LTL based framework augmented with model variables. Before describing the algorithm, the notion of pre-diagnosability of a system in LTL framework is explained.

5.3.2.1 Notion of Pre-diagnosability and Diagnosability :

First the system to be diagnosed for faults is formally modeled as M_d. M_d is given by an eight tuple:

$$M_d = (S_i, S_{i0}, V, \Sigma, \Im, R, AP, L) \text{ where,}$$

- S_i is the finite set of states.
- S_{i0} is the set of initial states, $S_{i0} \subseteq S_i$.
- V is the set of model variables.
- Σ is the set of events. Note that an event can be observable or unobservable depending on whether its occurrence can be perceived by available sensors.
- $\Im$ is the set of transitions. A transition $\tau \in \Im$ is defined as a three-valued tuple $\langle \sigma, check(v), assign(v) \rangle$, where $\sigma \in \{\Sigma \cup \epsilon\}$ is an event (on which the transition is fired), $check(v)$ is a boolean conjunction of equalities of a subset of model variables V, $assign(v)$ is a subset of model variables and assignments with values from their corresponding domain.
- $R \subseteq S_i \times \Im \times S_i$ is the transition relation.
- AP is finite set atomic propositions.
- L is a labeling function $S_i \rightarrow 2^{AP}$ that label each state with the set of atomic proposition true at that state.

Definition 5.4 (Transition Trace): *Let $s_0, s_1, ...$ be a state trace. A finite or infinite sequence of transitions $\{t_1, t_2, ...\}$ is called an **transition trace** if for every $i \in \{1, 2, ...\}$, s_{i-1}, s_i is part of the state trace and s_{i-1} moves to state s_i via a transition $t_i \in \Im$.*

The language generated by M_d, defined as a set of transition traces, is denoted by $lang(M_d) \subseteq R^*$. For every generated transition trace, a corresponding state trace can be found in the model. Note that state trace can be finite or infinite. As LTL framework works only on infinite state traces, a system model can be made non-terminating by adding

self-loops with $\langle \epsilon, -, - \rangle$ transition to any state $s \in S_i$ that does not have any outgoing transitions.

The transitions executed by M_d are filtered through an observation mask M. M takes a transition as input and generates its corresponding observable transition as output. The notion of observability is associated only with the triggering event of a transition, not with model variables. This is because any behavior measured from the system may be unobservable, but model variables are stored in the system memory so their values are always observable. Therefore the mapping of M is as follows:
$\langle \sigma, check(v), assign(v) \rangle \rightarrow \langle \sigma_m, check(v), assign(v) \rangle$,
where $\sigma \in \{\Sigma \cup \epsilon\}$, $\sigma_m \in \{\Delta \cup \epsilon\}$ and Δ is the set of observable events corresponding to Σ. Note that the observed event for ϵ is ϵ.

Definition 5.5 (observable equivalent transition): *Two transitions* $t_1 = \langle \sigma_1, check_1(v), assign_1(v) \rangle$ *and* $t_2 = \langle \sigma_2, check_2(v), assign_2(v) \rangle$ *are said to be observable equivalent if* σ_1 *and* σ_2 *produces the same observable event from* Δ, $check_1(v) \equiv check_2(v)$ *(i.e., same equalities over the same subset of model variables) and* $assign_1(v) \equiv assign_2(v)$ *(i.e., same subset of model variables with same assignments). This is represented as* $M(t_1) = M(t_2)$.

An infinite state trace is said to be faulty if it does not satisfy the LTL specification i.e., it violates the specification. A system model is faulty if it generates such a faulty path from its start state. In fault diagnosis, all such occurrences of faulty infinite paths are detected in finite delay as an instance of model checking problem. To be able to detect a fault within finite delay, a system must have an indicator as defined below.

Definition 5.6 (Indicator): *An indicator* (Ind_p) *is a finite state trace whose all infinite extensions are faulty.*

Definition 5.7 (pre-diagnosable): *A system model* M_d *is said to be pre-diagnosable w.r.t a specification* f *if its every faulty state trace possess an indicator as its prefix.*

Note that when pre-diagnosability of a system is tested, the observation limitation is withheld. It is assumed that all events are observable and verified whether under this condition, the occurrence of a fault can be deduced within a finite delay after its occurrence.

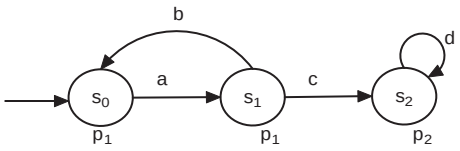

Figure 5.1: Example of Pre-diagnosability

The absence of an indicator for any faulty path implies that no matter how long the path is observed (even withholding the observability constraint i.e., considering that unobservable events are also being traced), it will never be possible to assure the occurrence of fault. So if the model is not pre-diagnosable it can never be diagnosable as further restriction is imposed thereby not allowing unobservable events to be monitored. Thus, if a system fails in pre-diagnosability test, detector building procedure can be discarded.

Now the idea of pre-diagnosis is shown with an example. The model shown in Figure 5.1 has states s_0, s_1, s_2 labeled by propositions p_1, p_2 and the transitions are a, b, c, d. The three tuple break up of the transitions is not shown in figure to avoid complexity. The model is pre-diagnosable with respect to the specification $f = Gp_1$ since the faulty state trace is $((s_0, s_1)^k, s_2^\omega)$, where $(s_0, s_1)^k$ denotes visiting state s_0 and s_1 a finite number of times i.e., $k \geq 1$ and ω denotes visiting a state any number of times starting $\omega \geq 1$. State s_2 is labeled with the proposition p_2. So once the system reaches s_2, Gp_1 is not satisfied there and trace leading to state s_2 becomes a faulty trace. This trace has $((s_0, s_1)^k, s_2)$ as its indicator. All infinite extensions of this indicator are faulty because once the system reaches state s_2, fault permanently remains.

Another important point to be remembered is that, a particular system model can be pre-diagnosable with respect to one specification whereas with respect to another it may not be so. The same system model of Figure 5.1 is not pre-diagnosable for the specification $f' = \mathbb{F}p_2$. The faulty state trace is $(s_0, s_1)^\omega$. An extension of it is the state trace $(s_0, s_1)^k, s_2$ which is non-faulty. Thus the faulty trace does not contain any indicator. The diagnosability of DESs in the setting of LTL is defined in Definition 5.8.

Definition 5.8 (diagnosable): *M_d is said to be diagnosable with respect to a specification f if the execution of any indicator in the system can be deduced within a finite delay from the observed behavior through the observation mask M.*

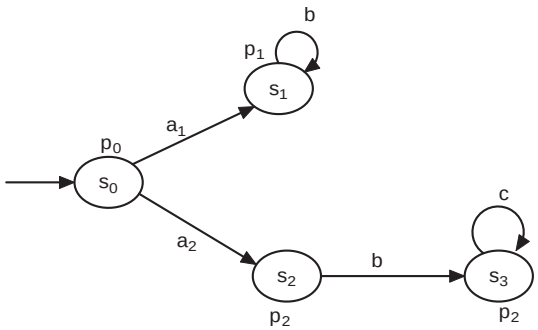

Figure 5.2: Example of Diagnosability

That means while testing diagnosability the observation restriction is applied and then it is verified whether the occurrence of a fault can be deduced within a finite delay. Idea of diagnosability is explained with an example system shown in Figure 5.2. The mask function for the system is defined as follows:

$M(a_1) = M(a_2) = a$ that means the transitions a_1 and a_2 are observable equivalent; $M(b) = b$; $M(c) = c$.

Now the system is diagnosable w.r.t specification $f = Gp_1$ since the faulty trace is (s_0, s_2, s_3^ω) and its corresponding indicator is (s_0, s_2, s_3). The observable transitions of the indicator is a, b, c and it renders the system to be diagnosable.

However a system being pre-diagnosable with respect to a formula never assures that it is also diagnosable. For example, the system shown in Figure 5.2 is pre-diagnosable with respect to a specification $f' = \mathbb{F}p_2$, as indicator is (s_0, s_1) exists for the faulty state trace (s_0, s_1^ω). However same model is not diagnosable with respect to the specification f' as observable equivalent transitions for this indicator is ab whose all infinite extensions are not faulty as the non-faulty state trace (s_0, s_2, s_3) has the same observation equivalence.

Once a system is found to be diagnosable, a detector for the same can be constructed. The detector detects the occurrence of each indicator in the system by observing transition traces generated by the system through the mask M.

5.3.2.2 Algorithm for Testing Diagnosability and Building the Detector

Now the algorithm to construct a finite state detector for failure diagnosis of system M_d w.r.t a specification f is discussed.

The LTL based failure diagnosis problem can be defined as follows:

Given a system model M_d, its observation mask M and an LTL specification f, test whether M_d is diagnosable and if so then construct a detector to detect whether an indicator has occurred by observing the behavior of M_d through the mask M.

The construction procedure of the detector evolves through four steps:

1. In the first step, the non-faulty behavior of the system is represented using a FSA called Buchi Automata, which is obtained automatically from LTL specification. An input sequence is accepted by the Buchi automata if there exists a run of the automaton which visits (at least) one of the final states infinitely often. As already stated, LTL provides a more intuitive approach where normal behavior of a system, given by a specification say f can be represented. Then from the specification a non-deterministic generalized automaton called Buchi automaton B_f is built automatically that accepts all infinite proposition traces over AP satisfying f. So it may be noted that unlike state based DES, where the non-faulty state-transition model to be generated manually, in LTL framework the corresponding model is generated automatically. B_f is defined as:

$$B_f = \left(C_f, \Sigma_{AP}, R_f, q_0^f, Fi\right), \text{ where}$$

   - C_f = A finite set of states.
   - $\Sigma_{AP} = 2^{AP}$ is the event set.
   - $R_f \subseteq C_f \times \Sigma_{AP} \times C_f$ is the transition relation.
   - q_0^f = the initial state.
   - $Fi \subseteq 2^{C_f}$ is the generalized Buchi acceptance condition i.e., the final states of the automata.

2. The next step is to test pre-diagnosability of M_d. In order to relax the observability restriction, propositions instead of events are taken under consideration. A FSA T_1 is constructed as the proposition synchronization of B_f and M_d. T_1 is defined as follows:

$$T_1 = (Q_1, \Sigma, \Im, R_1, Q_0^1, AP \cup Fi, L_1)$$

   where,

- $Q_1 = C_f \times S_i$ is the set of states.
- Σ is the set of events.
- $\Im$ is the set of transitions. This set is same as that of M_d.
- $R_1 \subseteq Q_1 \times \Im \times Q_1$ is the transition relation,
  $R_1 = \{\left(\left(c_f, s\right), t, \left(c'_f, s'\right)\right) \in Q_1 \times \Im \times Q_1 |$
  $\left(c_f, L\left(s'\right), c'_f\right) \in R_f, (s, t, s') \in R\}$. This formal notion implies that, if there is a transition t in M_d from state s to s' and a transition in Buchi B_f form state c_f to c'_f labeled with the proposition that is true at state s', then transition t can be added in T_1 that will take state (c_f, s) to (c'_f, s').
- $Q_1^0 = \{(c_f, s) \in Q_1 | (q_0^f, L(s), c_f) \in R_f, s \in S_{i0}\}$ is the set of initial states.
- $AP \cup Fi$ is the set of atomic propositions.
- $L_1 : Q_1 \rightarrow 2^{AP \cup Fi}$ is the labeling function such that $\forall \left(c_f, s\right) \in Q_1, fi \in Fi, p \in AP$: $\{fi \in L_1\left(c_f, s\right) \Leftrightarrow c_f \in fi\} \wedge \{p \in L_1\left(c_f, s\right) \Leftrightarrow p \in L\left(s\right)\}$. Let (c_f, s) be a state in T_1. It is labeled proposition that is true at state s of system model M_d. Additionally it may be labeled with fi implying that it is a final state in T_1, iff c_f is a final state in B_f having $L(c_f) = fi$.

Then from T_1 delete each state $q \in Q_1$ and its associated transitions if either q has no successor, or no state labeled with some $fi \in Fi$ can be reached from q; repeat this process until no more states and transitions can be deleted.

If every infinite proposition trace generated by T_1 visits the Fi-labeled states infinitely often then the model is said to be pre-diagnosable. Thus T_1 should satisfy the LTL formula $GFfi$.

3. In this step the diagnosability of M_d is tested. For that construct

$$T_2 = (Q_2, \Delta, \Im_M, R_2, Q_0^2)$$

as **projection of** T_1 **through M**. Thus if the language generated by T_1 is $lang(T_1)$ and that of T_2 is $lang(T_2)$, then $lang(T_2) = M(lang(T_1))$.

- $Q_2 = Q_0^1 \cup \{q \in Q_1 | \exists (q', t, q) \in R_1$, such that triggering event of $t \neq \epsilon\}$ is the set of states. Thus in Q_2, all the initial states of T_1 and states with transitions triggered by observable events leading into them, are included.
- Δ is the set of observable events.
- $\mathfrak{I}_M$ is the set of observable equivalent transitions of $\mathfrak{I}$.
- $R_2 \subseteq Q_2 \times \mathfrak{I}_M \times Q_2$ is the set of transitions. $\forall (q, t_M, q') \in Q_2 \times \mathfrak{I}_M \times Q_2, (q, t_M, q') \in R_2$ iff there exists a path $(q_0, t_1, q_1, ..., t_k, q_k, t_{k+1}, q_{k+1})\ (k \geq 0)$ in T_1 such that $(q_i, t_{i+1}, q_{i+1}) \in R_1$ for $0 \leq i \leq k$, $q_0 = q$, $q_{k+1} = q'$, triggering of t_i is ϵ for $1 \leq i \leq k$ and $M(t_{k+1}) = t_M$. This formal notion implies that a transition t_M from q to q' is added to T_2 if and only if there is a path in T_1 from q to q' in which all the transitions except the last one are triggered by unobservable events and the observable equivalent of the last transition is t_M.
- $Q_0^2 = Q_0^1$ is the set of initial states.

Then construct

$$T_2' = (Q_2, \Sigma, \mathfrak{I}, R_2', Q_0^2)$$

that accepts $M^{-1}(lang(T_2)) = M^{-1}M(lang(T_1))$. The transition relation $R_2' \subseteq Q_2 \times \mathfrak{I} \times Q_2$ is given as: $\forall (q, t, q') \in Q_2 \times \mathfrak{I} \times Q_2$ such that
$(q, t, q') \in R_2' \Leftrightarrow \{(q, M(t), q') \in R_2\} \vee$
$\{(q = q') \wedge (triggering\ event\ of\ t = \epsilon)\}$.
Thus the language generated by T_2' is all unobservable extension of observable equivalent language generated by T_1.

Then from T_2', construct T_3 as the **event synchronization of** T_2' **and** M_d.

$$T_3 = (Q_3, \Sigma, \mathfrak{I}, R_3, Q_0^3, AP, L_3),$$

where,

- $Q_3 = Q_2 \times S_i$ is the set of states.

- Σ is the set of events.
- $\Im$ is the set of transitions.
- $R_3 \subseteq Q_3 \times \Im \times Q_3$ is the transition relation such that $\forall ((q_1, s_1), t, (q_2, s_2)) \in Q_3 \times \Im \times Q_3$:
  $((q_1, s_1), t, (q_2, s_2)) \in R_3 \Leftrightarrow \{(triggering\ event\ of\ t \neq \epsilon) \wedge$
  $\left((q_1, t, q_2) \in R_2'\right) \wedge ((s_1, t, s_2) \in R)\} \vee$
  $\{(triggering\ event\ of\ t = \epsilon) \wedge (q_1 = q_2) \wedge ((s_1, t, s_2) \in R)\}$.
  This implies that all transitions that are common in T_2' and M_d, are included in T_3.
- $Q_0^3 = Q_0^2 \times S_{i0}$ is the set of initial states.
- AP is the set of atomic propositions.
- $L_3 : Q_3 \rightarrow 2^{AP}$ is the labeling function such that $\forall (q, s) \in Q_3,\ L_3(q, s) = L(s)$.
- Delete each state $q \in Q_3$ and its associated transitions if q has no successor.

The language accepted by $T_3 = lang(T_2') \cap lang(M_d)$. Thus T_3 generates the traces of M_d that share an observation with traces in T_1 and thus all traces of T_3 must themselves be non-faulty. Note that here the observable restriction is added for verifying diagnosability. So to confirm diagnosability, check whether every infinite proposition trace generated by T_3 satisfies f using LTL model checking methods i.e T_3 must satisfy the LTL formula f.

4. This final step is for construction of the detector. Output (T_2, M_{T_2}) as the detector D. $M_{T_2} : \Im_M^* \rightarrow \{fault\}$ is a partial function such as: $\forall s \in \Im_M^*, M_{T_2}(s) = fault$ if $s \notin lang(T_2)$.
   The detector observes transition traces generated by M_d through the mask M. If a transition trace s is not in the language generated by T_2 then the detector outputs 'fault'.

The detector for NDP attacks is built in Section 5.4 using the above mentioned algorithm. Thus in the process of building the detector, all the above steps are shown in execution

with the example of NDP attacks.

Note: The diagnosability test and detector design procedure as given in [43] has polynomial complexity to the number of system states. Above mentioned procedure only adds model variable assignments and checking that can be accomplished in constant time. Thus the complexity remains polynomial.

5.3.3 Proving Correctness of a Specification

As discussed in the sub-section 5.3.2.2, the first step in the "Algorithm for Testing Diagnosability and Building the Detector" is the non-faulty representing of the system using a FSA called Buchi Automata, which is obtained automatically from LTL specification; this in an automated procedure and not prone to errors. However, the LTL specification is manually developed from the English language specification. Although the LTL framework provides a user-friendly paradigm to represent English language specifications to LTL formula, still this being an manual process may have errors. In this sub-section we discuss a procedure to prove correctness of the LTL formula (developed from a English language specification) using satisfiability; the scheme is proposed in [71] and is discussed here in brief.

5.3.3.1 Decomposition Rules

Let the specification (i.e., an LTL formula) to be validated be sp. Checking the satisfiability of the specification ascertains whether the formula is correct. To verify that it must be checked whether there exists a truth assignment to the sub-formulas of the specification such that the overall specification is true. If no such truth assignment is found then it can be inferred that the formula is not true under any circumstances. Hence the formula is declared as an incorrect one. Thus a specification is first broken into sub- formulas called basic formulas. Basic formulas are either atomic propositions sp or are of the form Xsp. Two identities are used for breaking a specification sp into basic formulas:

- $Gsp \equiv (sp \vee XGsp)$ i.e., a specification sp is said to be true globally if it is true in the current state and in the next state Gsp holds. It is a recursive definition which implies that sp is true in every state.

- $sp_1 \cup sp_2 \equiv (sp_2 \vee (sp_1 \wedge X(sp_1 \cup sp_2)))$ - i.e., $sp1$ until $sp2$ can be recursively defined

as $sp1$ being true in current state and in the next state ($sp1 \cup sp2$) holds. The base condition is $sp2$ becoming true.

Now using these two identities a set of decomposition rules is obtained. These rules are as follows:

- $\neg\neg sp \equiv \{sp\}$
- $\neg X sp \equiv \{X\neg sp\}$
- $sp_1 \wedge sp_2 \equiv \{sp_1,\ sp_2\}$
- $\neg\left(sp_1 \wedge sp_2\right) \equiv \{\{\neg sp_1\}, \{\neg sp_2\}\}$
- $Fsp \equiv \{\{sp\}, \{XFsp\}\}$
- $Gsp \equiv \{\{sp, XGsp\}\}$
- $sp_1 \cup sp_2 \equiv \{\{sp_2\}, \{sp_1, X\left(sp_1 \cup sp_2\right)\}\}$
- $\neg\left(sp_1 \cup sp_2\right) \equiv \{\neg sp_2, \neg sp_1 \vee \neg X\left(sp_1 \cup sp_2\right)\}$

To convert an non-elementary formula to an elementary formula, a reduced tableau is constructed. Reduced tableau is a graph and its construction proceeds by using these aforesaid rules.

5.3.3.2 Construction of Reduced Tableau

The algorithm for reduced tableau construction is as follows:

1. First a node is created and labeled by sp, where sp is the formula to be tested. Following that steps 2 and 3 are repeatedly applied. These steps creates children nodes. If a node n is needed to be created and labeled by a set of formulas F_n, first it is checked whether there already exists a node in the graph labeled by F_n. If such a node is found then an edge is created from n to that node, otherwise a new node labeled by $T_n \neq F_n$ say, is created.

2. For a node n which is labeled by F_n and F_n contains an non-elementary formula f, create a child of n labeled by $(F_n - \{f\}) \cup S_i \cup \{f^*\}$ where decomposition rule for f

is $f \equiv \{S_i\}$ and f^* denotes marked f. To avoid repeating over same formula during construction, the formulas are marked with $*$ for which decomposition rules have already been applied.

3. For a node n that is labeled by only elementary and marked formulas, create a child of n labeled by X formulas with their outermost X removed.

A **state** is a node that contains only elementary or marked formulas. A **pre-state** is a node that is either a immediate child of a state or the initial node of the graph. Once the graph is formed it is reduced by removing the unsatisfiable nodes. The elimination rules are as follows:

1. A node is unsatisfiable if it contains both proposition p and its negation $\neg p$. Such type of nodes are eliminated first.

2. A node is eliminated if all of its successor nodes are eliminated.

3. Eliminate a pre-state that contains a formula of the form Gsp or $sp1 \cup sp2$ or Fsp and the formula is not satisfiable, where satisfiability is defined as:
   A formula Gsp_2 or $sp_1 \cup sp_2$ or Fsp_2 is satisfiable in a pre-state if there is a path in reduced tableau leading from the pre-state to a node containing the formula sp_2.

From the reduced tableau we can determine whether a specification is satisfiable using the following definition.

Definition 5.9 (satisfiable specification): *A specification is called satisfiable if the initial node of its corresponding reduced tableau is not eliminated, otherwise the specification is said to be unsatisfiable.*

Now we explain the above theory using a simple example.

5.3.3.3 An Example

Now the correctness of a sample LTL specification is validated using the procedure discussed above. Let the specification be $sp = (G\neg p \wedge Fp)$l it means that proposition p must be false in every state (i.e., $\neg p$ is true) and at the same time in some future state p must be true. Clearly this is a contradictory statement that cannot be true. This can be proved formally by using the reduced tableau of Figure 5.3.

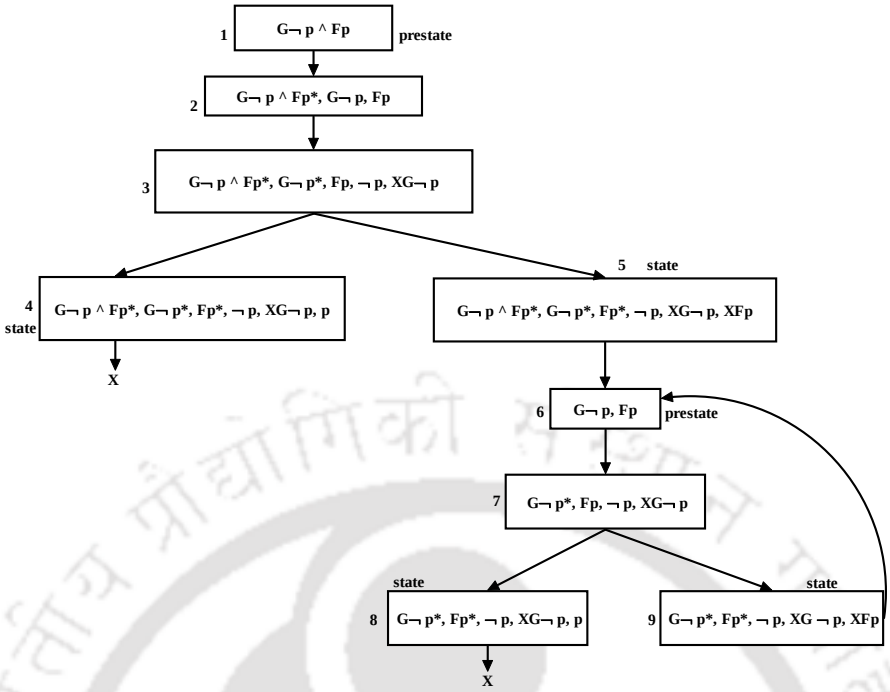

Figure 5.3: Reduced tableau for $sp = (G\neg p \wedge Fp)$

Node 1 is the prestate having the formula $(G\neg p \wedge Fp)$. First the main operator $\wedge$ is selected and node 2 is generated as per the decomposition rule $G\neg p \wedge F \equiv \{G\neg p, Fp\}$; in node 2 formula $(G\neg p \wedge Fp)$ is marked with * and $G\neg p$, Fp are added. Node 3 is created by processing $G\neg p$ as $\neg p, XG\neg p$. Now, node 4 and node 5 are created by processing Fp as $\{p\}, \{XFp\}$, respectively. Node 4 and node 5 are states because they contain only elementary and marked formulas. Node 6 is created by applying the rule "create a child of n labeled by X formulas with their outermost X removed". Node 4 is to be deleted as it contains both p and $\neg p$. Node 7 is created by processing $G\neg p$. Finally, node 8 and node 9 are cerated by processing Fp. It may be noted that node 8 and node 9 are states. Node 8 is also eliminated as it contains both p and $\neg p$. Now node 6 is also deleted (following rule 3) as node 6 contains Fp and no path from node 6 goes to a node containing p (as node 8 is already deleted). In this way eventually initial node 1 will get deleted. This proves that the LTL specification is unsatisfiable according to Definition 9.

5.4 IDS for NDP attacks Using LTL based DES Framework

In this section the application of the adapted LTL based DES theory for detection of NDP attacks is presented. Starting with assumptions, this section discusses the modeling of NDP attacks using LTL based DES framework, construction of the detector (which is the IDS) and checking its correctness. Also, the basic motivation of augmenting the LTL based DES framework [43] with model variables and active probing is illustrated in the context of modeling the NDP attacks.

5.4.1 Assumptions

The proposed scheme relies on the following assumptions regarding IPv6 LAN.

1. All nodes are IPv6 configured using Stateless address autoconfiguration (SLAAC) mechanism or have been assigned static IP.
2. Genuine non-compromised nodes on the link which are expected to reply a Neighbor Solicitation message (either unicast or multicast) must do so within a specific time interval t_{req}.
3. IDS is a trusted machine with a static IP-MAC binding. It has two network interfaces dedicated to their respective purposes; one being responsible to collect network data in the LAN through port mirroring and the other being exclusively used for handling NS or NA probes requests/replies.
4. IDS has static information of the $IP - MAC$ association of interfaces of gateway.

As NDP is stateless, the NDP packet-sequences under normal and attack condition is same. For example, under spoofing attack condition, when an NA packet is received by host A from attacker D having IP(B)MAC(D), A updates its cache with IPMAC pair as IP(B)MAC(D). It may be noted that under normal condition i.e., A receives the NA packet from B having IP(B)MAC(B), A updates its cache with IPMAC pair as IP(B)MAC(B). So, under attack or normal condition there is no change in the sequence of NDP packets. Similar holds for all NDP related attack; this will be elaborated for each attack case in the next sub-section. In the proposed scheme active probing technique is used to create a difference in sequence of NDP packets under normal and attack condition, which is modeled using

DES framework and LTL specification is written for the sequence (of packets) normal condition of the protocol. The DES detector (which is the IDS) monitors NDP packet sequence in the network and raises an alarm in case of violation of the sequence.

The basic architecture of the LAN being monitored using the IDS is shown in Figure 5.4. As shown in the figure, the IDS comprises a DES detector and a packet handler. The packet handler receives all traffic due to port mirroring and sends probe requests to verify genuineness of NDP packets. It generates events (namely NS packet received, probe sent etc.), which are monitored passively by the detector. The detector checks if the events adhere to the LTL specification and declares attack or normal condition.

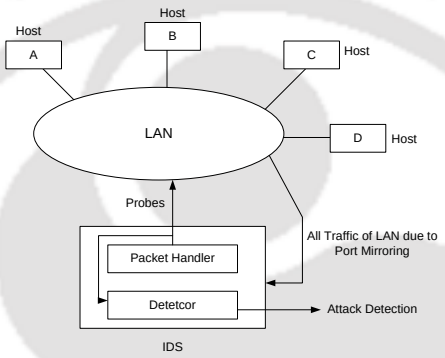

Figure 5.4: Basic Architecture of LAN with IDS

In the next sub-section, we elaborate the packet handler component of the IDS.

5.4.2 Packet handler of the IDS

As discussed before, the proposed IDS uses probes to create a difference in sequence of NDP packets under normal and attack condition. To assist in the probing and separating the genuine NDP packets with that of spoofed ones, we maintain information related to packets being verified, probes being sent and probe responses, in some data tables. In this sub-section we discuss the format and purpose of these tables followed by the probing algorithms, which together comprise the packet handler.

Henceforth in the discussion, the following short notations are used:
IPS - Source IP Address; *IPD* - Destination IP Address; *IPT* - Target IP Address; *MACS* - Source MAC Address; *MACD* - Destination MAC Address; *NSP* - Neighbor Solicitation packet; *NAP* - Neighbor Advertisement packet. Source IP of *NSP* is denoted as NSP_{IPS}; similar subscripting will be used to denote destination IP, source MAC and destination

MAC for *NSP* and *NAP*; Fields of any table would be represented by $\langle TableName \rangle_{\langle field \rangle}$; e.g., $AUTHT_{IPS}$ represents the source IPv6 address field of "Authenticated Table".

The format and basic requirement of these tables are enumerated below.

1. **Authenticated Bindings Table:** Every time an NS or NA packet is verified to be genuine an entry is made in this table, denoted as *AUTHT*. It has three fields, source IP–$AUTHT_{IPS}$, source MAC–$AUTHT_{MACS}$, and time when the solicitation/ advertisement packet is received by IDS– $AUTHT_{Ti}$.

2. **Spoofed Bindings Table:** For spoofed solicitations/ advertisements the details are entered in this table denoted as *SPOOFT*. It has three fields, source IP–$SPOOFT_{IPS}$, source MAC–$SPOOFT_{MACS}$, and time when the solicitation/advertisement packet is received by IDS–$SPOOFT_{Ti}$.

3. **DAD Table:** Every time an NS or NA packet related to DAD is received or sent the details are recorded in DAD table. An NS packet which has *IPS* as ::(unspecified) and an NA packet which has the solicitation flag set are related to DAD. DAD table has four fields, target IP–$DADT_{IPT}$, source MAC–$DADT_{MACS}$, time when the packet is received by IDS–$DADT_{timeofreceipt}$ and time of sending the (NS) probe –$DADT_{timeofsending}$. Some of the fields may also be kept blank.

We need to limit the history of the data that we store in the tables, so all entries older than some t_{flush} time are flushed from tables. Timestamps in the tables are used for logging and also to assist in flushing the stale data.

The probing technique has four main modules namely, NS-HANDLER(), NA-HANDLER(), EXPHAND() and DTDHAND(). These are elaborated in Algorithm 5.1, Algorithm 5.2, Algorithm 5.3 and Algorithm 5.4, respectively. Now we elaborate these modules as follows. Algorithm 5.1 process NS packets in the network and generates events correspondingly. These event sequences are monitored by the DES detector for verifying attck/normal condition. It is assumed that the events generated comprise all NDP header related information e.g., *PRNSP* (i.e., probe for *NSP*) comprises *IPS* from which the *NSP* has arrived, which is being probed. The following are the different cases of *NSP* and the events generated by Algorithm 5.1.

- *Malformed:* For any NS packet *NSP*, the algorithm first verifies if it is malformed (i.e., any changes in the immutable fields of the packet header). Algorithm 5.1 simply

exits (by setting a Status flag) on such NS packets. The packet details are entered into *SPOOFT*.
Motivation: Detection of such abnormal cases are trivial (based on a single *NSP*) and is not considered in DES modeling and attack detection.

- *Probe solicitation from IDS:* If the packet is not malformed but a solicitation packet from (IDS) i.e., NSP_{IPS} is IP of IDS and NSP_{MACS} is MAC of IDS, Algorithm 5.1 skips processing of this packet; no probe needs to generated from IDS as we assume that IP-MAC pairing of the IDS is known and validated.

- *Source IP of NS packet is unspecified:* If *NSP* is not from IDS, it is checked whether the source IP of the NS packet is unspecified. NSP_{IPS} =:: signifies a DAD. There can be three sub-cases as follows.

  – An *NSP* is received, whose target IP is already present in *DADT* and a corresponding probe is sent for that packet within t_{req} time. *DADT* table is used to verify this sub-case by checking if there has been a *DAD* for the same target IP and if solicitation probe has already been sent for attack detection. In this sub-case *PRNSPRCVDAD* event is generated by the algorithm. Informally speaking, *PRNSPRCVDAD* stands for Probe Response *NSP* received for DAD.
  Motivation: This reflects the fact that attacker is performing a DAD attack on victim by vying for the same target *IP* as that of victim. It may be noted that the target IP used by the IDS in the probe is random but among one that is in use. Such a target IP can be determined by choosing a random IP from *AUTHT*, because *AUTHT* comprises *IP* and *MAC* addresses of hosts which are in use and known to be genuine. So if a *DAD* probe is sent by the IDS whose target *IP* is already in use, then only in an attack situation we receive another *DAD NSP* vying for the same target *IP*. In the genuine situation we will receive a *DAD NAP* stating that the target *IP* is under use. The DES detector will use the event *PRNSPRCVDAD* (along with other event sequence) to determine attack nor normal case.

  – Otherwise, if there is an entry in *DADT* table (within t_{req}), but probe has not been sent for it within t_{req} time, event *NSPDAD*2 is generated. This means that the *NSP* (*DAD* packet) has same target *IP* as that of a previously received *NSP*,

denoted as *NSPDAD*1 (details of *NSPDAD*1 is given in next sub-case). Thus, this new *NSP* packet is named as *NSPDADS*2 (as *NSPDAD*1 and *NSPDAD*2 have same target *IP* and only received at different timings). A random IP IP_{rand} is selected from the *AUTHT* table and a solicitation DAD probe is sent as if $NSPDAD1_{MACS}$ is performing a *DAD* for an alternative (random) IP. The details (i.e., $\langle IP_{rand}, NSPDAD1_{MACS}, -, timeofsending\rangle$) of sending the probe is added to *DADT*, event *PRNSPDAD* is generated and EXPHAND() is called passing the random IP as parameter.

Motivation: This probe is sent to check the presence of an attacker who is performing *DAD* attack. It may be noted that in this sub-case two *NS DAD* packets have been received with same target *IP* address. As a *DAD* probe is sent with random but existing *IP* address as target and the source MAC is the MAC address of the first DAD *NSP* (i.e., *NSPDADS*1), the attacker cannot guest whether it is a probe or a real *DAD* request (where the host is trying to configure an alternative IP address). To make the attack persistent the attacker will either (i) respond to the probe stating that it also uses the same target address (response is *DAD NSP*) or (ii) it wants to configure the same target address (response is *DAD NAP*, to be discussed in Algorithm 5.2). In case of (ii) the genuine host (using the target IP under question) will also respond with an *DAD NAP*. So if a single *NSP* response or two *NAP* responses to the probe are received within t_{req} (checked by module EXPHAND()), attack can be detected.

– Again an *NSP* for DAD can be received, whose target IP is not present in *DADT*. If there has been no previous *DAD* for NSP_{IPT}, the details of the *NSP* packet is added to the *DADT* table, event *NSPDAD*1 is generated and EXPHAND() is invoked. This *NSP* is therefore a first one with a particular target *IP* and source address as unspecified. Hence, the name *NSPDAD*1.

Motivation: The current *NSP* is the first (i.e., only one at present) *DAD* packet vying for a target *IP*; generally, this is a genuine event when a host boots up or changes its *IP* address and informs it all hosts in the group. So no probe is sent and only a record (i.e., $\langle NSP_{IPT}, NSP_{MACS}, timeofreceipt, -\rangle$) is kept in *DADT* and EXPHAND() is executed to check if other related *DAD* packets arrive within t_{req}.

- *NSP* is not for the above cases, i.e., a simple NS packet: DTDHAND() is invoked passing the source $IP - MAC$ of the NS packet as parameters. DTDHAND() returns true if $NSP_{IPS} - NSP_{MACS}$ pair is already verified to be *authenticated* or *spoofed*. If it returns true, events *NSPRCV* (informally saying, *NSPRCV* stands for *NSP* received) and *DTD* (informally saying, *DTD* stands for already determined using tables and probing not required) are generated. If it returns false, only event *NSPRCV* is generated. NS probe request is multicast from the IDS asking for the *MAC* address of NSP_{IPS}, event *PRNSP* (informally saying, *PRNSP* stands for probe *NSP*) is informed and EXPHAND() is invoked.

  Motivation: In an attack scenario there will be two replies, one from genuine host (having the NSP_{IPS}) and the other from the attacker. The reply from the genuine host will have mismatching $IP - MAC$ pair compared to the *NSP* under question. Thus active probing will make difference in sequence of packets for normal vs attack condition, which can be used for detection by the DES detector.

All these steps of the algorithm will be illustrated using examples (in coming subsections), when the attacks will be modeled using LTL based DES framework.

Algorithm 5.2 process NA packets in the network and generates events correspondingly. The following are the different cases of *NAP* and the events generated by Algorithm 5.2.

1. Malformed: For any NA packet *NAP*, the algorithm first finds if it is malformed, similar to that done for an *NS* packet, and sets the status accordingly.

2. *NAP* is for the IDS: Next, it verifies whether the NA packet is destined for IDS. This can be determined if NAP_{IPD} is IP of IDS and NAP_{MACD} is MAC of IDS. These NA packets are responses to the probes sent by IDS. They can be a response either to *PRNSPUPDAD* (this probe is sent to test reachability of a host in case of DAD attack detection) or to *PRNSP* probe (sent for the verification of a IP-MAC pair for any *NS* or *NA* packet). The details of probe *PRNSPUPDAD* will be elaborated later, and *PRNSP* is discussed in Algorithm 5.1. A tables named *DADT* is maintained which keep record of the sent *PRNSPUPDAD* packets. Therefore these *NA* packets destined to IDS can be handled as per the three subcases given below.

   (a) Response to *PRNSPUPDAD*: The *NAP* is first checked to verify whether there

Algorithm 5.1 *NS* Handler

Input : *NSP* - Neighbor Solicitation packet.
Output: Events- *PRNSPRCVDAD*, *PRNSPDAD*, *NSPDAD*1, *NSPDAD*2, *NSP*, *DTD* and *PRNSP*.
Update in *AUTHT*, *SPOOFT* and *DADT*.

if *NSP* is *malformed* **then**
  status ← *abnormal*;
  add {NSP_{IPS}, NSP_{MACS}, NSP_{IPD},
  NSP_{MACD}, time of receipt} in *SPOOFT*;
  Exit
else if NSP_{IPS} == *IP*(*IDS*) and NSP_{MACS} == *MAC*(*IDS*) **then** /**NSP* is for the IDS*/
  Exit
else if NSP_{IPS}== :: (unspecified address) **then**
  if NSP_{IPT} == $DADT_{IPT}[i]$ AND time of receipt of *NSP* - $DADT_{timeofsending}[i] < t_{req}$, for some i, $1 \leq i \leq DADT_{MAX}$ **then**
/* Denotes the case when attacker replies with *NSP* for the probe *PRNSPDAD* sent by IDS. The target IP of *NSP* is already present in *DADT* and it is received within t_{req} time of sending the probe */
    Generate event *PRNSPRCVDAD*;

  else if NSP_{IPT} == $DADT_{IPT}[i]$ AND time of receipt of *NSP* - $DADT_{timeofreceipt}[i] < t_{req}$, for some i, $1 \leq i \leq DADT_{MAX}$ **then**
/* Denotes the case when two *NSP* are received vying for same *IP* within t_{req} time */
    Generate event *NSPDAD*2;
    Select IP_{rand} as a random IP from *AUTHT* table (and MAC_{rand} as MAC of IP_{rand});
    send NS probe request *PRNSPDAD* from IDS with $PRNSPDAD_{IPS}$ = :: and $PRNSPDAD_{MACS}$ = $DADT_{MACS}[i]$ and $PRNSPDAD_{IPT}$ =IP_{rand};
    add ⟨IP_{rand}, $DADT_{MACS}[i]$, −, time of sending⟩ to *DADT*;
    Generate event *PRNSPDAD*;
    call *EXPHAND*(IP_{rand});
    Exit

  else/* First *DAD NSP* i.e., target IP is not present in *DADT* */
    Generate event *NSPDAD*1;
    add ⟨NSP_{IPT}, NSP_{MACS}, time of receipt, −,⟩ in *DADT*;
    Call expiry handler *EXPHAND*(NSP_{IPT}, NSP_{MACS});
  end if

end if

is an entry in *DADT* table made in last t_{req} time such that $NAP_{IPS} = DADT_{IPT}$. If such an entry is found, then the *NAP* is a response to *PRNSPUPDAD*; event *PRNAPUPDAD* is generated. A random IP IP_{rand} is selected from the *AUTHT* table and a solicitation DAD probe is sent as if $NSPDAD1_{MACS}$ is performing a *DAD* for an alternative (random) IP. The details are added to the *DADT* and event *PRNSPDAD* is generated.

Motivation: In case of DAD attack, first it is verified whether the machine that claims to use said *IP* is actually up; *PRNSPUPDAD* stands for probe *NS* packet (for checking if host is)"up" for DAD. A *PRNAPUPDAD* in response to a *PRNSPUPDAD* conforms that the machine is indeed up. If no such response arrives then the algorithm can exit (detecting attack) straightaway saving lot of computation.

(b) Response to *PRNSP*: If none of the cases are satisfied, the algorithm intimates the receipt of *PRNAP*, which is a response to *PRNSP*.

Motivation: The response to *PRNSP*s will be used by the DES detector to detect attack. In an attack scenario there will be two *NAP*s–the reply from the genuine host will have different $IP - MAC$ pair compared to the corresponding *NSP* for which the *PRNSP* is sent, while the response from attacker will have same $IP - MAC$ pair. In a genuine case, there will be one *NAP* with same $IP - MAC$ pair as that of the corresponding *NSP*.

3. *NAP* is not for the IDS and solicitation flag is set: Algorithm 5.2 checks if the solicitation flag of the packet is set. If found so, it means that this *NAP* arrives in response to a previously sent *NS* packet. This can be a case for DAD.

   Then we check if *NAP* is multicast. If yes, it signifies that it is a DAD packet. Now, it can be a reply generated by an existing user to a previous *NS* DAD packet *NSPDAD*1 sent by an entering node or it can be a reply to the probe *PRNSPDAD*.

   If the source IP of the *NA* DAD packet matches with any of the target IPs in the *DADT* table (say, i^{th} entry) and its time of receipt in within t_{req} of "time of receipt" of the i^{th} entry, event *NAPDAD* is generated; this *NA* DAD packet is the reply to *NSPDAD*1. Event *NAPDAD* is generated. It is checked whether $NAP_{IPS} - NAP_{MACS}$ is already verified to be genuine or spoofed, using the DTDHAND(). If DTDHAND() returns

true, event *DTD* is generated. If it returns false, an unicast NS probe *PRNSPUPDAD* is sent by IDS to verify the reachability of $NAP_{IPS} - NAP_{MACS}$. The source IP-MAC of *PRNSPUPDAD* is that of the IDS and the destination IP is NAP_{IPS}. In *DADT* $\langle NAP_{IPS}, MAC_{IDS}, -, timeofsending \rangle$ is recorded. However, if the field "time of receipt" of the i^{th} entry is blank, then the field "time of sending" is checked. If time of arrival of the *NA* DAD packet is within t_{req} from "time of sending" of the i^{th} entry, event *PRNAPDAD* is generated; this *NA* DAD packet is the reply to *PRNSPDAD*. Event *PRNAPDAD* is generated.

Motivation: In a network when a new host wants to enter, it first checks whether the *IP* that it wants to take is already in use. The *NSPDAD*1 packet is sent from it to verify this. Now if there is a host who uses that *IP* replies. The reply *NAPDAD* is a broadcast message and its solicitation flag is also set. This can also be spoofed by attacker to launch *DAD* attack. So whenever IDS finds such packet it runs the detection process. In the detection process the IDS first checks if *NAPDAD* is a reply to *NSPDAD*1. If so, it is checked if the genuineness of IP-MAC pair of *NAPDAD* is already validated (by looking into *AUTHT*/*SPOOFT*). If they are not yet validated, it is checked if the machine that claims to use the said *IP* (i.e., $NAPDAD_{IPS}$) is actually up; sends a *PRSNPUPDAD* probe to check reachability of $NAPDAD_{IPS}$. As discussed in step 2(a), if response arrives, DAD probe *PRNSPDAD* is sent.

If the *NAPDAD* is not a response to *NSPDAD*1 then it is checked if it is a reply to some DAD probe (*PRNSPDAD*) sent earlier. If so, an event *PRNAPDAD* is generated. As discussed, in normal scenario there would one reply (*PRNAPDAD*) to *PRNSPDAD*, while in the case of attack there would be two replies. So by counting the number of *PRNAPDAD* events attack can be detected.

4. General *NAP*, i.e., neither destined to IDS or response to any probe However, if solicitation flag of the *NA* packet is not set, it is a general *NAP* packet. First, *DTDHAND*() is called to check whether its $IP - MAC$ pair is already verified. If so, *DTD* is intimated, else NS probe *PRNSP* is sent.

   Motivation: As in the case of a general *NAP*, in this case also a probe *PRNSP* is sent. In an attack scenario there will be two replies, one from genuine host and the other from the attacker. The reply from the genuine host will have different $IP - MAC$ pair compared to the *NAP* for which the probe is sent under question, while the response

from attacker will have same $IP - MAC$ pair. In case of genuine condition only one response will arrive having same $IP - MAC$ pair as in the NAP.

Algorithm 5.2 *NA* Handler

Input : *NAP* - Neighbor Advertisement packet.
Output: Events *NAP*, *PRNAP*, *PRNSP*, *NAPDAD*, *PRNAPDAD*, *PRNAPUPDAD*, *PRNSPUPDAD*, *PRNSPDAD*, *DTD* and Update in *AUTHT*, *SPOOFT*, *DADT*

if *NAP* is *malformed* **then**
  status ← *abnormal*;
  add {NAP_{IPS}, NAP_{MACS}, NAP_{IPD}, NAP_{MACD}, time of receipt} in *SPOOFT*; Exit;
else if NAP_{IPD} == *IP*(*IDS*) and NAP_{MACD} == *MAC*(*IDS*) **then** /*NSP sent to IDS*/
  if NAP_{IPS} == $DADT_{IPT}[i]$ AND time of receipt of *NAP* - $DADT_{timeofsending}[i] < t_{req}$, for some i, $1 \leq i \leq DADT_{MAX}$ **then** /* Denotes receipt of response for the reachability probe *PRNSPUPDAD* sent by *NA* handler in t_{req} time*/
    Generate event *PRNAPUPDAD*;
    Select IP_{rand} as a random IP from *AUTHT* table (and MAC_{rand} as MAC of IP_{rand});
    send NS probe request *PRNSPDAD* from IDS with $PRNSPDAD_{IPS}$ =:: and $PRNSPDAD_{MACS}$ = $DADT_{MACS}[i]$ and $PRNSPDAD_{IPT}$ = IP_{rand};
    add $\langle IP_{rand}, DADT_{MACS}[i], timeofsending, -\rangle$ to *DADT*;
    Generate event *PRNSPDAD* and call *EXPHAND*(IP_{rand});
  else/* Receipt of response for normal *NS* probe *PRNSP* sent from *NS/NA* handler*/
    Generate event *PRNAP*. Exit;
  end if
else if *NAP* solicitation flag is set **then** /*NA packet may be for DAD*/
  if NAP_{IPD}== all node multi-cast address **then** /* Denotes a DAD packet*/
    if NAP_{IPS} == $DADT_{IPT}[i]$ AND time of receipt of *NAP* - $DADT_{timeofreceipt}[i] < t_{req}$, for some i, $1 \leq i \leq DADT_{MAX}$ **then** /* Denotes the case when an existing user claims to be using the target IP*/
      Intimate *NAPDAD*;
      if *DTDHAND* (NAP_{IPS}, NAP_{MACS}) **then** /* If already verified*/
        Generate events *NAPDAD* and *DTD*; Exit;
      else/* Reachability is tested by sending a unicast probe *PRNSPUP*/
        Send unicast NS probe request to NAP_{IPS};
        Generate event *PRNSPUPDAD*;
        add $\langle NAP_{IPS}, -, timeofsending, -\rangle$ to *DADT* and call *EXPHAND*(NAP_{IPS}, NAP_{MACS}); Exit;
      end if
    else*NAP*$_{IPS}$ == $DADT_{IPT}[i]$ AND time of receipt of *NAP* - $DADT_{timeofsending}[i] < t_{req}$, for some i, $1 \leq i \leq DADT_{MAX}$ /* Denotes that case when a response to *PRNSP* arrives*/
      Generate event *PRNAPDAD*;
    end if
  end if
else/* Denotes receipt of normal *NA* packet*/
  Generate event *NAP*;
  if *DTDHAND* (NAP_{IPS}, NAP_{MACS}) **then** /* If already verified*/
    Generate event *DTD*; Exit;
  else/* IP-MAC pair of the NA packet is tested by sending probe *PRNSP* */
    send NS probe request *PRNSP* to NAP_{IPS} from IDS;
    Generate event *PRNSP* and call *EXPHAND*(NAP_{IPS}, NAP_{MACS}); Exit;
  end if
end if

Algorithm 3 describes DTDHAND() handler. The job of the handler is to check whether the $IP - MAC$ pair in question is genuine (return TRUE) or spoofed (return FALSE) by verified from *AUTHT* or *SPOOFT* table, i.e, without sending probe DTDHAND() verified IP-MAC pairs. It takes a host's IP and MAC denoted as *CIP* and *CMAC* as input. *CIP* is searched in the Authenticated table. If a match is found as $AUTHT_{IPS}[i]$ and the corresponding source MAC address $AUTHT_{MACS}[i]$ in the table is same as *CMAC*, the packet has genuine IP-MAC pair which is already determined by the detector. This genuineness is obtained without explicit use of the DES detector. Also, NS probes are not sent in this case thereby saving some NDP traffic. *CIP*, *CMAC*, and time of receipt are stored in the Authenticated table. In case of a match in *CIP* and mismatch in the *CMAC* (i.e., $CIP = AUTHT_{IPS}[i]$ and $CMAC \neq AUTHT_{MACS}[i]$) the packet is detected to be spoofed *without explicit use of the DES detector and no NS probes are sent.* The details of the packet are stored in the Spoofed table. On the other hand if *CIP* is not found in *AUTHT* (in field $AUTHT_{IPS}$), then both *CIP* and *CMAC* are searched in *SPOOFT* (i.e., $CIP = SPOOFT_{IPS}[i]$ and $CMAC = SPOOFT_{MACS}[i]$). If a match is found then it implies that the IP-MAC pair is already detected to be spoofed. No probes are sent and details of the packet are added *SPOOFT*. Thus DTDHAND() checks whether the $IP - MAC$ pair can be verified from either of the *AUTHT* or *SPOOFT* table. If so then it returns *TRUE*, otherwise it returns *FALSE*.

Algorithm 4 describes EXPHAND(). It takes a host's IP as input. A clock is triggered once the handler is called. Once the clock ticks for t_{req} amount of time, event *EXP* is intimated and the IP address that is received as parameter is returned. EXPHAND() is used as a watchdog timer for probes sent to a host (IP) to wait for responses till t_{req}. After t_{req} time all responses for the probe (identified by the host IP to which the probe is sent) are discarded. As already mentioned in the assumptions (subsection 5.4.1), genuine hosts reply to reply a NS message within a specific time interval t_{req}.

In the next subsections, we will discuss LTL based DES modeling, detector design and verifying its concreteness for different NDP related attacks.

5.4.3 Neighbor Solicitation Spoofing attack

5.4.3.1 LTL based DES Modeling of NS spoofing

In this subsection we will consider *NS* spoofing attack, discussed in Subsection 5.2.1.1.

Algorithm 5.3 *DTDHAND* Handler

Input : IP and MAC address of a host.
Output: Updated *AUTHT* or *SPOOFT* table and a return value of *TRUE* or *FALSE*.

assign $CIP \leftarrow$ IP address that is passed as parameter;
assign $CMAC \leftarrow$ MAC address that is passed as second parameter;
if $CIP == AUTHT_{IPS}[i]$ AND $CMAC == AUTHT_{MACS}[i]$, for some i, $1 \leq i \leq AUTHT_{MAX}$ **then**
  "Status is Genuine";
  add {*CIP*, *CMAC*, time of receipt} in *AUTHT*;
  Return *TRUE*;
  Exit
else if $CIP == AUTHT_{IPS}[i]$ AND $CMAC \neq AUTHT_{MACS}[i]$, for some i, $1 \leq i \leq AUTHT_{MAX}$ **then**
  "Status is Spoofed";
  add {*CIP*, *CMAC*, time of receipt} in *SPOOFT*;
  Return *TRUE*;
  Exit
else if $CIP == SPOOFT_{IPS}[i]$ AND $CMAC == SPOOFT_{MACS}[i]$, for some i, $1 \leq i \leq SPOOFT_{MAX}$ **then**
  "Status is Spoofed";
  add {*CIP*, *CMAC*, time of receipt} in *SPOOFT*;
  Return *TRUE*;
  Exit
else
  Return *FALSE*;
  Exit;
end if

Algorithm 5.4 *EXPHAND* handler

Input : IP address of a host.
Output: Generate event *EXP* and return *EIP*.

assign $EIP \leftarrow$ IP address that is passed as parameter;
assign $t \leftarrow currenttime$;
while *true* **do**
  clock ticks up by one step;
  if $currenttime - t == t_{req}$ (value of t_{req} is predefined) **then**
    Generate event *EXP*;
    return *EIP*;
  end if
end while

The components of the DES model of NDP are as follows.

Model Variables:

A set of model variables $IPS, MACS$ is required by the model. These variables are used to identify the IP and MAC of the machine whose solicitation or advertisement is currently being processed.

Domain of IPS is $\{d.d.d.d.d.d.d.d \mid d \geq 1 \text{ and } d \leq 65535\}$.

Domain of $MACS$ is $\{d.d.d.d.d.d \mid d \geq 1 \text{ and } d \leq 255\}$.

Event Set:

The following are the events used in the DES modeling. They are either generated directly or derived from the algorithms discussed above.

- $NSPRCV$ denotes the receipt of Neighbor solicitation packet (Generated by Algorithm 5.1).
- DTD occurs when the IP/MAC pairing of the received NS/NA packet is already present in Authenticated/Spoofed table (Generated by Algorithm 5.1 and Algorithm 5.2).
- $PRNSP$ denotes sending of the NS probe request (Generated by Algorithm 5.1 and Algorithm 5.2, when it needs to verify authenticity of $IP - MAC$ pair of general NA or NS packets).
- $PRNAP$ Denotes the receipt of response to $PRNSP$ (Generated by Algorithm 5.2)
- EXP Whenever an NS probe is sent, EXPHAND() is invoked for that corresponding IPS. Once t_{req} time interval expires, EXPHAND() intimates EXP event. This event is used to determine if the NA probe responses arrive within t_{req} time of sending the corresponding $PRNSP$.
- u is an unobservable event.

Transition

A transition is a three-valued tuple $\langle \sigma, check(v), assign(v) \rangle$ as defined in Subsection 5.3.2. For example $\langle NSP, -, IPS \leftarrow NSP_{IPS} \,\&\, MACS \leftarrow NSP_{MACS} \rangle$ is a transition that event NSP triggers with assignment of NSP_{IPS} to IPS

and NSP_{MACS} to $MACS$. There is no checking condition in this example transition. $\langle PRNSP, IPS == PRNSP_{IPD}, - \rangle$ is a transition that event $PRNSP$ triggers with checking if IPS is same as $PRNSP_{IPD}$. There is no assignment in this case.

Proposition Set

The following are the set of propositions:

p_1 = *Detector is in its initial state.*

p_2 = *NS packet is obtained.* This is true after transition triggered by NSP event occurs.

p_3 = *NS probe request has been sent.* This is true after transition triggered by $PRNSP$ event occurs.

p_4 = *NA probe response is received having* $NAP_{IPS} - NAP_{MACS}$ *same as that of* $IPS - MACS$. This is true after transition $\langle PRNAP, -, IPS == NSP_{IPS}$ &
$MACS == NSP_{MACS} \rangle$

p_5 = *NA probe response is received having* $NAP_{IPS} - NAP_{MACS}$ *different from that of* $IPS - MACS$. This is true after transition $\langle PRNAP, -, IPS == NSP_{IPS}$ &
$MACS \neq NSP_{MACS} \rangle$

$p_6 = t_{req}$ *expires or*
IP/MAC pairing of the received NS packet is already present in Authenticated/Spoofed table. This is true after transitions triggered either by event DTD or event EXP occurs.

Mask Function

As already assumed, the IDS receives all NDP packets (solicitations, advertisements, solicitation probes and probe advertisements) in the network due to port mirroring. So corresponding events, NSP, NAP, $PRNAP$, $PRNSP$ are measurable. Also, events generated by EXPHAND() and DTDHAND() i.e. EXP and DTD are observable. Finally, model variables are also observable. So Mask Function is as follows:

$M(NSP) = NSP, M(PRNSP) = PRNSP,$
$M(PRNAP) = PRNAP, M(EXP) = EXP,$
$M(DTD) = DTD, M(u) = \epsilon.$

The system DES model M_d i.e., representation of NDP related packets under probing for normal and NS spoofing cases is shown in Figure 5.5.

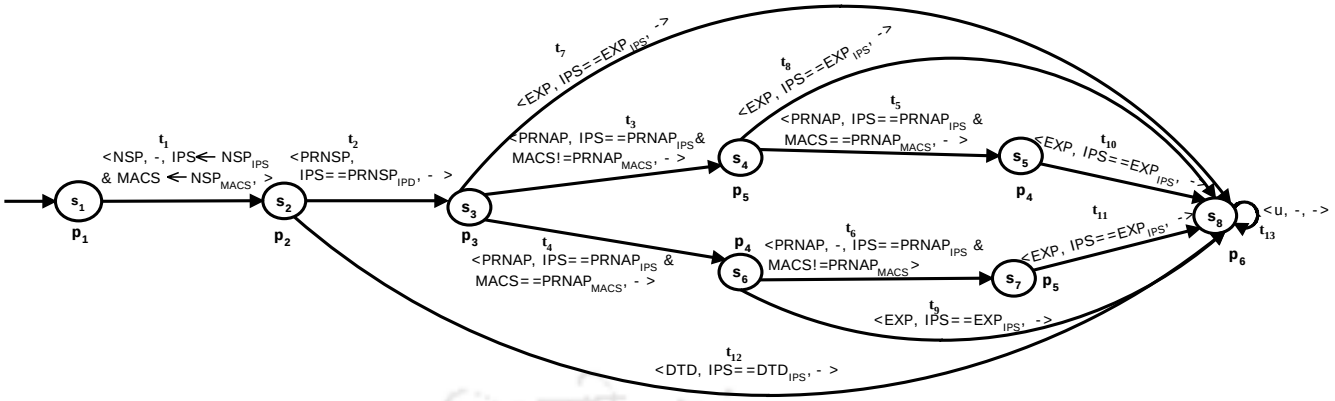

Figure 5.5: System Model of *NS* Spoofing

$M_d = (S_i, S_{i0}, V, \Sigma, \Im, R, AP, L)$ where,

- $S_i = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$.
- $S_{i0} = \{s_1\}$.
- $V = \{IPS, MACS\}$.
- $\Sigma = \{NSP, PRNSP, DTD, PRNAP, EXP, u\}$.
- $\Im = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}\}$. [1]
- R is the transition relation shown in Figure 5.5.
- $AP = \{p_1, p_2, p_3, p_4, p_5, p_6\}$.
- L is the labeling function shown in Figure 5.5.

An overview of the DES model for the normal/ attack scenario is explained next.

Under normal conditions, on receipt of an *NS* packet, event *NSP* triggers the transition t_1 from state s_1 to s_2. The model variables *IPS* and *MACS* are assigned with source IP address and source MAC address of the *NS* packet; this is shown in transition t_1. At initial state, proposition p_1 holds and at state s_2, proposition p_2 holds. If the $IP - MAC$ pair is already present in the Authenticated Table, t_{12} takes the system to s_8 via event *DTD* and matching *IPS* with DTD_{IPS}. Otherwise, an NS Probe Request (i.e., *PRNSP* packet) is sent

[1] In the transitions of DES modeling (in Figure 5.5), the events, checking of model variables and assignment of model variables have been shown explicitly. However, for simplicity henceforth, t_is shall be used to denote the transitions.

and event *PRNSP* triggers transition t_2 from state s_2 to state s_3. For the corresponding *IPS*, EXPHAND() is invoked by Algorithm 5.1 to keep track of the t_{req} time interval. Under normal condition, one single *NA* packet (i.e., *PRNAP* packet) arrives in response to the NS Probe packet within t_{req} time whose $IP-MAC$ pair matches with $PRNAP_{IPS}-PRNAP_{MACS}$. The system moves to state s_6 where p_4 holds. It may be noted that the response to the *PRNSP* under question is identified by matching $PRNAP_{IPS}$ with *IPS*; this is performed in transition t_4 which is triggered by event *PRNAP* and model variables *IPS* and *MACS* should match $PRNAP_{IPS}$ and $PRNAP_{MAC}$, respectively. The system waits at s_6 until *EXP* event is triggered. The correct instance of the clock is identified by matching EXP_{IPS} with *IPS*. The system terminates at s_8. To make the system infinite a self loop "$u,-,-$" (an unobservable event) has been added to s_8.

Now the model under *NS* spoofing and probe *PRNSP* is discussed. As under normal condition, event *NSP* triggers transition t_1 from state s_1 to s_2. The model variables *IPS* and *MACS* are assigned to source IP address and source MAC address of the *NS* packet. If the $IP-MAC$ pair is already present in the Spoofed Table t_{12} takes the system to s_8 via event *DTD*. Otherwise, NS Probe Request is sent and event *PRNSP* triggers transition t_2 to state s_3. At s_3 there are three options-

(i) No *NA* packet arrives in response within t_{req} time (as *IPS* may be non existent). Event *EXP* causes transition t_7 to s_8;

(ii) Probe Response (i.e., *PRNAP*) from normal host arrives, whose $IP-MAC$ pair is different from that of $NSP_{IPS}-NSP_{MACS}$. As $NSP_{IPS}-NSP_{MACS}$ is spoofed, the $IP-MAC$ pair in probe response from genuine host will be different. Event *PRNAP* causes transition t_3 to go to state s_4;

(iii) NA Probe Response (*PRNAP*) from attacker arrives, which of course will have $IP-MAC$ pair same as that of $NSP_{IPS}-NSP_{MACS}$. Event *PRNAP* causes transition t_4 to go to state s_6. Note that model cannot distinguish which reply is from attacker and which one is from genuine host. It merely denotes the fact that there are two responses received, one with matching $IP-MAC$ pair and the other with a mismatch.

At s_4 there can be two options-

(i) No other NA probe response arrives within t_{req} time as it may be the case that attacker does not send any reply to the probe. Event *EXP* causes transition t_8 to s_8;

(ii) NA Probe Response from attacker with $IP-MAC$ pair same as that of $NSP_{IPS}-NSP_{MACS}$

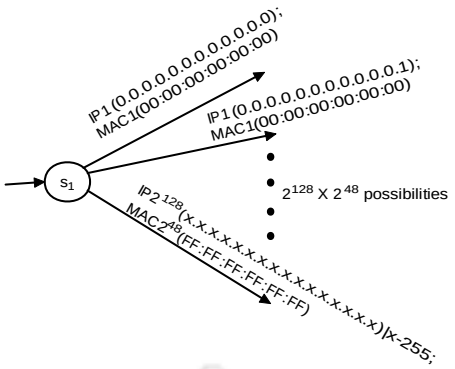

Figure 5.6: State Explosion in System Model

arrives and transition t_5 occurs and system goes to state s_5.
Similar options exist for state s_6-
(i) No other NA probe response arrives within t_{req} time as it may be the case that genuine host does not send any reply to the probe (as *IPS* may be non existent). Event *EXP* causes transition t_9 to go to state s_8;
(ii) NA Probe Response from genuine host with mismatched $IP - MAC$ compared to $NSP_{IPS} - NSP_{MACS}$ arrives and transition t_6 occurs and system goes to state s_7.
The system waits at s_5 or s_7 until *EXP* event is triggered. The system terminates at s_8.
Note: If model variables had not been used then there would have been $2^{128} \times 2^{48}$ transitions (and states) from s_1, each representing the IP-MAC pair sent by the request packet. As the attacker can send any arbitrary IP-MAC pair, we need all such combinations, each represented by a state. This would lead to state explosion problem as shown in Figure 5.6.

5.4.3.2 Proof of Correctness of the LTL Specification: Normal condition for *NS* packets under probing

Now, we develop the LTL based specification for normal condition of *NSP* under probing. The specification in natural language in terms of the events is given as:
Neither any probe response with non-matching $IP - MAC$ pair to that of the original *NSP* packet should be encountered nor should the *PRNSP* probe request have no probe response arriving for it, until the system terminates.

So in terms of the transition set, the specification can be given as:
Neither t_3 caused by $\langle PRNAP, -, IPS == PRNAP_{IPS}$

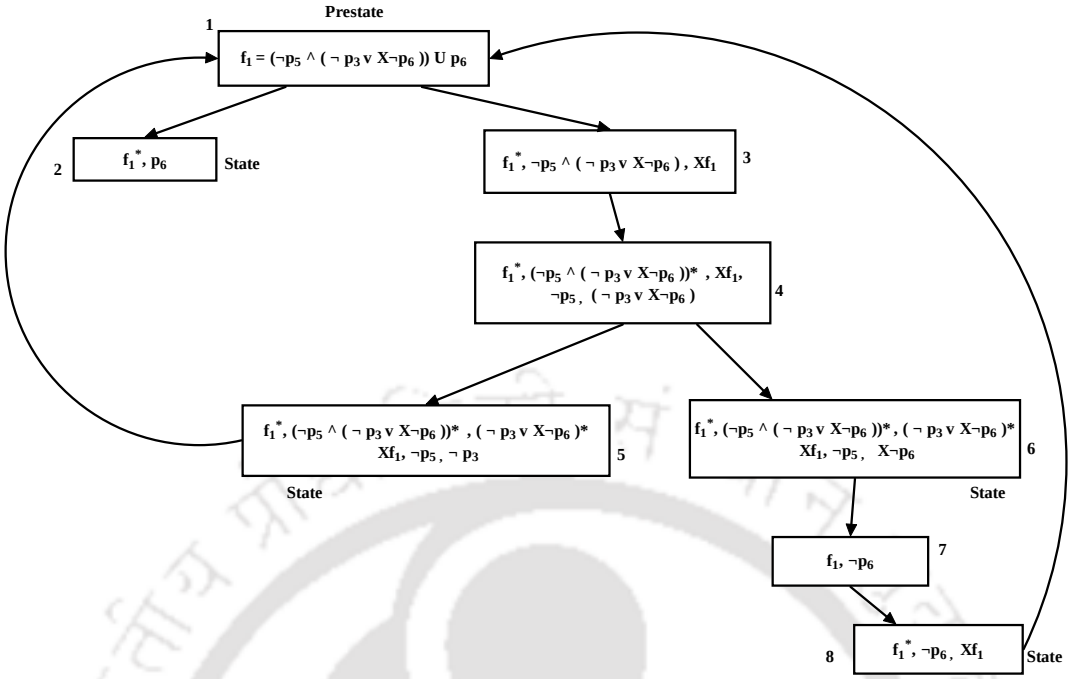

Figure 5.7: Reduced Tableau of f_1

& $MACS \neq PRNAP_{MACS}\rangle$ should be encountered nor should a t_7 caused by EXP occur immediately after transition t_2 until the system terminates.

Now in terms of the proposition set, the specification can be given as:
Neither p_5 should be encountered nor should a p_6 hold immediately after transition p_3 until the system terminates (at state s_8 where p_6 holds). Thus, the above specification is given by the LTL formula (in terms of propositions) f_1 defined as:
$(\neg p_5 \wedge (p_3 \rightarrow X\neg p_6)) \cup p_6$.

Before going into the diagnoser construction process, it must be shown that the specification is a correct one. As already discussed, in order to verify the correctness, a reduced tableau of the specification is built. The reduced tableau of f_1 is shown in Figure 5.7, built using the algorithm described in Subsection 5.3. The $\rightarrow$ in $f1$ replaced with $\wedge$ so that tableau can be constructed using rule set given in Subsection 5.3.3.1– $(\neg p_5 \wedge (\neg p_3 \vee X\neg p_6)) \cup p_6$.

First the main operator $\cup$ is selected and the initial node is split into two nodes (node 2 and node 3) using the decomposition rule: $(\neg p_5 \wedge (\neg p_3 \vee X\neg p_6)) \cup p_6 \equiv$
$\{p_6\}, \{\neg p_5 \wedge (\neg p_3 \vee X\neg p_6)\, , X((\neg p_5 \wedge (\neg p_3 \vee X\neg p_6)) \cup p_6)\}$ and the initial formula is marked. Initial formula is represented as f_1 in figure. Node 2 contains proposition p_6 and marked

formula f_1^*; so is nothing to be done on the state. In next step, node 3 is processed with respect to operator $\wedge$ and node 4 is obtained (by using the rule $(\neg p_5 \wedge (\neg p_3 \vee X\neg p_6)) \equiv \{\neg p_5, (\neg p_3 \vee X\neg p_6)\}$. Rest of the construction process can be explained in a similar way using the rule sets given in Subsection 5.3.3.1.

It can be seen that the initial state of the reduced tableau does not get eliminated. As we already know, from Definition 9 that if initial state is not deleted then the specification is correct. This proves that f_1 is a correct specification.

5.4.3.3 Construction of DES detector for NS spoofing

In this subsection we discuss the process of constructing the DES detector for *NS* spoofing using the steps discussed in Section 5.3.

Formation of Buchi Automata

The fist step is construction of Buchi automata for the specification f_1, which is as follows:

$$B_{f_1} = \left(C_{f_1}, \Sigma_{AP}, R_{f_1}, q_0^{f_1}, Fi\right).$$

which is shown in Figure 5.8 where,

- $C_{f_1} = \{B_0, B_1, B_2, B_3, B_4, B_5, B_6\}$ is the set of states.
- $\Sigma_{AP} = \{p_1, p_2, p_3, p_4, p_6\}$ is the event set that causes transition from one state to another.
- R_{f_1} is the transition relation, where $R_{f_1} \subseteq C_{f_1} \times \Sigma_{AP} \times C_{f_1}$.
- $q_0^{f_1} = B_0$ is the initial state.
- $Fi = \{B_3, B_6\}$ are the F_1 labeled final states where the specification is satisfied.

The Buchi automata for f_1 is shown in Figure 5.8. It can be seen from the Figure that B_f has three paths emanating from the initial state B_0. All the three paths satisfy $\neg p_5$. Also, states that are reached with proposition p_3, like B_2 and B_4 do not have transitions with p_6 outgoing from them. This satisfies $(p_3 \rightarrow X\neg p_6)$ part of f_1. Once state B_3 is reached, f_1 is satisfied and thus any proposition coming after that leads to another final state B_6. B_f is a non-deterministic automata and there are multiple paths that can be taken with the same set of propositions. It can be verified manually that B_f accepts all proposition traces over AP satisfying f_1.

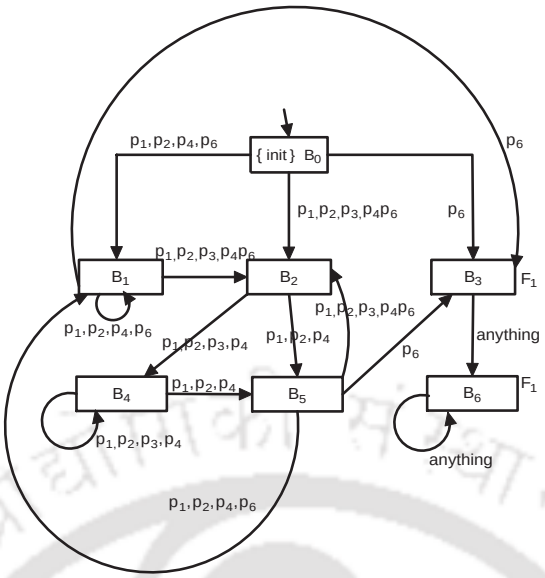

Figure 5.8: Buchi Automata T_{f1} for f_1

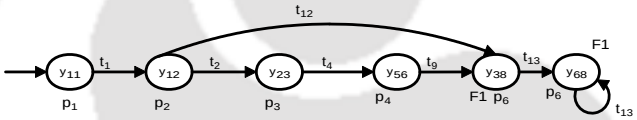

Figure 5.9: Model T_1 for f_1

Proposition Synchronization of Buchi automata and the Model for test of pre-diagnosability

The pre-diagnosability test requires a proposition synchronization of M_d and B_f. This proposition synchronization T_1 is shown in Figure 5.9 where y_{ij} denotes $\left(B_i, s_j\right)$, $B_i \in C_{f_1}$ and $s_j \in S_i$. The proposition synchronization T_1 is derived as:

$$T_1 = (Q_1, \Sigma, \Im, R_1, Q_0^1, AP \cup Fi, L_1) \text{ where,}$$

- $Q_1 = \{y_{11}, y_{12}, y_{23}, y_{56}, y_{38}, y_{68}\}$ is the set of states.
- Σ = set of events which is same as that of the model M_d.
- $\Im = \{t_1, t_2, t_4, t_9, t_{12}, t_{13}\}$ is the set of transitions
- R_1 is the transition relation, which is illustrated in Figure 5.9.
- $Q_0^1 = \{y_{11}\}$.
- $AP \cup F1$ is the new set of proposition.

- L_1 is the labeling function, which is illustrated in Figure 5.9.

According to T_1 construction algorithm given in Subsection 5.3.2.2, $Q_1^0 = \{(c_{f_1}, s) \in Q_1 | (q_0^{f_1}, L(s), c_{f_1}) \in R_{f_1}, s \in S_{i0}\}$. Therefore $Q_0^1 = \{(B_1 s_1), (B_2 s_1)\}$ as $L_1(s_1) = p_1$, $(B_0, p_1, B_1) \in R_{f_1}$, $(B_0, p_1, B_2) \in R_{f_1}$ and $s_1 \in S_{i0}$. So, the initial states are $\{y_{11}, y_{21}\}$; shortly we will show that the second initial state y_{21} will get merged with y_{11}.

The few other initial steps of constructing T_1 are as follows.
From the initial state $y_{11}(= \{B_1, s_1\})$ through transition t_1 we go to state y_{12}, i.e., $((B_1, s_1), t_1, (B_1, s_2)) \in R_1$ as $L(s_2)$=p_2, $(B_1, p_2, B_1) \in R_{f_1}$ and $(s_1, t_1, s_2) \in R$. In other words, $(y_{11}, t_1, y_{12}) \in R_1$. y_{11} is labeled as p_1, i.e., $p_1 \in L_1(y_{11})$ as $p_1 \in L_1(s_1)$. y_{12} is labeled as p_2, i.e., $p_2 \in L_1(y_{12})$ as $p_2 \in L_1(s_2)$.
Also, from initial state y_{11} through transition t_1 we go to state y_{22}, i.e., $((B_1, s_1), t_1, (B_2, s_2)) \in R_1$ as $L(s_2)$=p_2, $(B_1, p_2, B_2) \in R_{f_1}$ and $(s_1, t_1, s_2) \in R$. y_{22} is also labeled as p_2. States y_{12} and y_{22} are merged as they are reached from same state y_{11} through same transition t_1 and have the same label p_2.

In a similar way, second initial state y_{21} gets merged with y_{11}, explained as follows. From the initial state $y_{21}(= \{B_2, s_1\})$ through transition t_1 we go to state y_{42}, i.e., $((B_2, s_1), t_1, (B_4, s_2)) \in R_1$ as $L(s_2)$=p_2, $(B_2, p_2, B_4) \in R_{f_1}$ and $(s_1, t_1, s_2) \in R$. In other words, $(y_{21}, t_1, y_{42}) \in R_1$. y_{21} is labeled as p_1, y_{42} is labeled as p_2. From the initial state y_{21} through transition t_1 we go to another state y_{52}, as $L(s_2)$=p_2, $(B_2, p_2, B_5) \in R_{f_1}$ and $(s_1, t_1, s_2) \in R$. States y_{42} and y_{52} can be merged; reason being similar to merging of y_{12} and y_{22}. Finally, states y_{12} and y_{42} are merged as they have same label p_2, they are reached from states y_{11} and y_{21} respectively, which have same label p_1, and through same transition t_1.

Finally states y_{11} and y_{21} are merged as they have same label p_1 and lead to states which have already been merged. After merging all such equivalent states the final T_1 is obtained, shown in Figure 5.9. As defined earlier if every infinite proposition generated by T_1 visits the *Fi* labeled states infinitely often, the system model is pre-diagnosable. Now in terms of LTL model checking it can be stated that T_1 must satisfy the LTL specification $G\mathbb{I}Fi$. Thus the pre-diagnosability checking reduces to LTL model checking that can be done by NuSMV model checker.

A Brief Description of NuSMV: NuSMV [66] provides a language for describing the models and it directly checks the validity of LTL formula(s) on these models. It takes as input a text consisting of a program describing a model and some specifications. It

```
MODULE main
VAR
 ev : {NSP,PRNSP,DTD,PRNAPONE,EXP,UNOBV};
 status : {pone,ptwo,psix,pfour,pthree};
ASSIGN
 init(ev) := NSP;
 init(status) := pone;
 next(status) := case
           (ev=NSP) : ptwo;
           (ev=PRNSP) : pthree;
           (ev=DTD) : psix;
           (ev=PRNAPONE) : pfour;
           (ev=EXP) : psix;
           TRUE : psix;
           esac;
 next(ev) := case
           (ev=NSP) : {DTD,PRNSP};
           (ev=PRNSP) : PRNAPONE;
           (ev=PRNAPONE) : EXP;
           TRUE : UNOBV;
           esac;
LTLSPEC
 G F (status=psix)
```

Figure 5.10: NuSMV Code snippet for testing pre-diagnosability of system

```
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>

*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

-- specification  G ( F status = psix)  is true
[itispris@cse bin]$
NuSMV-2.5.3-x86_64-redhat-linux-gnu/b : bash
```

Figure 5.11: Output showing that the system is prediagnosable

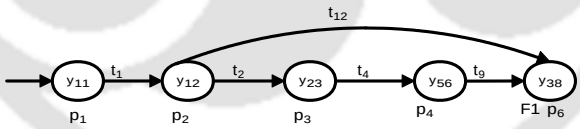

Figure 5.12: Model T_2 for f_1

produces as output either the word 'true' if the specification holds, or a trace where the specification is not satisfied. The code snippet for testing the pre-diagnosability of M_d is shown in Figure 5.10. Since the only F_1 labeled state in T_1 is where p_6 is true, testing for $G\mathbb{F}p_6$ suffices for the pre-diagnosability test. The corresponding output is shown in Figure 5.11.

Testing Diagnosability

As the system model passes the pre-diagnosability test, the next step (Subsection 5.3.2.2,

Step 3) is to test the diagnosability of the system with respect to the specification.

Synthesis of Masked T_1: Using the mask function M defined earlier T_2 is constructed as shown in Figure 5.12.

$$T_2 = (Q_2, \Delta, \Im_M, R_2, Q_0^2) \text{ where}$$

- $Q_2 = \{y_{11}, y_{12}, y_{23}, y_{56}, y_{38}\}$.
- $\Delta = \{NSP, PRNSP, PRNAP, EXP, DTD\}$.
- $\Im_M = \{t_{M1}, t_{M2}, t_{M4}, t_{M9}, t_{M12}\}$. t_{Mi} denotes the observable equivalent of transition t_i (of M_d).
- R_2 is the transition relation, shown in Figure 5.12.
- $Q_0^2 = \{y_{11}\}$.

According to the algorithm for T_2 construction in Subsection 5.3.2.2, Q_2 contains all those states of T_1 such that there is a transition triggered with an observable event leading into that state. So, y_{68} (which was in T_1) is eliminated as t_{13} is the only transition leading to y_{68} whose observed event is ϵ.

Unmasking T_1: Now from T_2, T_2' is constructed as shown in Figure 5.13 that accepts $M^{-1}(lang(T_2))$.

$$T_2' = (Q_2, \Sigma, \Im, R_2', Q_0^2) \text{ where}$$

- $Q_2 = \{y_{11}, y_{12}, y_{23}, y_{56}, y_{38}\}$.
- $\Sigma = \{NSP, PRNSP, PRNAP, EXP, DTD, u\}$.
- $\Im = \{t_1, t_2, t_4, t_9, t_{12}, t_{13}\}$.
- R_2', shown in Figure 5.13.
- $Q_0^2 = \{y_{11}\}$.

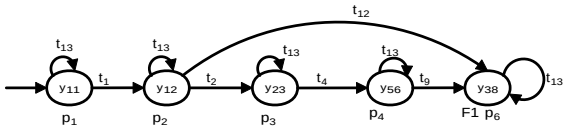

Figure 5.13: Model T_2' for f_1

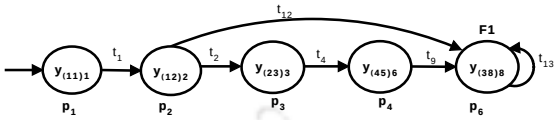

Figure 5.14: Model T_3 for f_1

Compared to T_2, in T_2' we add self loop transitions in all states e.g., $(y_{11}, t_{13}, y_{11}) \in R_2'$ as $y_{11} = y_{11}$ and
triggering event of t_{13} *is* ϵ.

Diagnosability Tested by Model Checking:
To test the diagnoability, T_3 is constructed from the event synchronization of T_2' and the system model as shown in Figure 5.14.

$$T_3 = (Q_3, \Sigma, \mathfrak{I}, R_3, Q_0^3, AP, L_3) \text{ where,}$$

- $Q_3 = \{y_{(11)1}, y_{(12)2}, y_{(23)3}, y_{(56)5}, y_{(38)8}\}$. States in Q_3 are represented as $y_{(lm)n}$, where $y_{(lm)} \in Q_2$ and $s_n \in S_i$.
- $\Sigma =$, same as M_d
- $\mathfrak{I} = \{t_1, t_2, t_4, t_9, t_{12}, t_{13}\}$.
- R_3, transition relation, shown in Figure 5.14.
- AP, is the proposition set, same as M_d.
- $L_3 = Q_3 \rightarrow 2^{AP}$ is the labeling function, shown in Figure 5.14.

As already discussed, T_3 generates traces of M_d that share an observation with traces in T_1. So, all the u triggered (self loop) transitions on states $y_{11}, y_{12}, y_{23}, y_{56}, y_{38}$ get eliminated as none of them belong to R (i.e., not present in M_d). All other transitions (from T_2') remain.

In this manner T_3 is obtained.
If M_d is diagnosable then every infinite proposition trace generated by T_3 must satisfy f_1. Again it is an LTL model checking problem that was solved using NuSMV model checker; the system was found to be diagnosable with respect to f_1.

Final Detector

As model is diagnosable with respect to the specification, its detector can be constructed. T_2 (Figure 5.12) along-with M_{T_2} forms the final detector where,

$$M_{T_2} : \Delta^* \rightarrow \{fault\} \text{ is a partial function defined as } \forall s \in \Delta^*, M_{T_2}(s) = \text{fault if } s \text{ is not generated by } T_2.$$

The detector observes the transition-traces generated by M_d through the mask M. If an observed transition-trace is not in the generated language of T_2 , then the detector outputs that a 'fault' has occurred. For example, if $t_1, t_2, t_3, t_5, t_{10}$ is a transition trace generated by M_d, its observation equivalent transition trace is $t_{M1}, t_{M2}, t_{M3}, t_{M5}, t_{M10}$ which is not generated by T_2. $t_{M1}, t_{M2}, t_{M3}, t_{M5}, t_{M10}$ corresponds to p_1, p_2, p_3, p_5, p_4 which violates $(\neg p_5 \wedge (p_3 \rightarrow X\neg p_6)) \cup p_6$. Thus, the detector outputs 'fault' for the trace. On the other hand, for t_1, t_2, t_4, t_9, whose observation equivalent transition trace is $t_{M1}, t_{M2}, t_{M4}, t_{M9}$ is generated by T_2.
$t_{M1}, t_{M2}, t_{M4}, t_{M9}$ corresponds to p_1, p_2, p_4, p_6 which adheres to $(\neg p_5 \wedge (p_3 \rightarrow X\neg p_6)) \cup p_6$. Thus, the detector outputs 'normal' for this case.

5.4.3.4 An Example to illustrate detection of *NS* spoofing

Let Figure 5.4 (subsection 5.4.1) represents the basic architecture of the LAN that is being monitored by DES based detector (i.e., IDS). It has five machines, *A*, *B*, *C*, *D* and *E*, among which *E* is the IDS and *D* is the attacker. Other machines viz *A*, *B*, *C* are genuine hosts. It is also assumed that there is no entry in either of *AUTHT* or *SPOOFT* tables regarding these five machines (i.e., startup situation). The following is the sequence of events:

- Packet 1: Attacker *D* sends (multicast) a spoofed NS packet (source IP address as IP(A) and source MAC address as MAC(D)), to query about the MAC address of *B*. As the NS packet is multicast, all hosts update their cache with IP(A) associated with MAC(D). As already discussed, on receipt of any NS packet the hosts update their

caches with the source IP-MAC pair. This leads to a situation where all traffic a host wants to send A would go to D. It may be noted that in a genuine situation where host A sends a NSP with $IP(A) - MAC(A)$ does not lead to change in sequence of NDP packets. In other words, in genuine situation all hosts update their cache with the IP-MAC pair received in the NSP (which is similar to attack situation *in terms of NDP packet sequence*). Now we will illustrate how probe $PRNSP$ leads to difference in sequence of NDP packets and attack detection.

NS_HANDLER() (Algorithm 5.1) receives Packet 1. The packet is not malformed, not sent by the IDS and source IP address is specified (i.e., not a response for DAD query). So, Algorithm 5.1 intimates event NSP and M_d moves to state s_2 via measured transition t_{M1}. The algorithm calls $DTDHAND(NSP_{IPS}, NSP_{MAC})$, which returns "false" as no entry corresponding to NSP_{IPS}, NSP_{MAC} can be found in $AUTHT$ or $SPOOFT$; initially $AUTHT$ or $SPOOFT$ are empty. So Algorithm 5.1 sends probe $PRNSP$ to NSP_{IPS}, event $PRNSP$ is generated (which causes M_d moves to state s_3 via measured transition t_{M2}) and finally $EXPHAND()$ is called.

- Packet 2: It may be noted that $PRNSP$ is nothing by a simple NSP in terms of NDP traffic. So packet 2 is $PRNSP$, which is received by NS_HANDLER() and it exits as it is a NSP from IDS.

- Packet 3 and packet 4: Within t_{req} time in response to $PRNSP$, let attacker respond with a NAP having $IP(A) - MAC(D)$ so as to present its original spoofed NS packet as genuine; let this be packet 3. Now, genuine host A will respond with another NAP to the probe with its genuine $IP(A) - MAC(A)$ pair (packet 4); in NDP all genuine hosts respond to NSP within t_{req}. These two NSPs (packets 3 and 4, respectively) will be handled by NS_HANDLER() (Algorithm 5.2) in the following way.

  Packet 3 is not malformed, sent to IDS but not present in $DADT$. So Algorithm 5.2 intimates event $PRNAP$ and M_d moves to state s_6 via measured transition t_{M4}. In state s_3 of M_d (Figure 5.5) enabling event of t_4 is $PRNAP$ with checking condition of model variables as $IPS == PRNAP_{IPS}$ and $MACS = PRNAP_{MACS}$ (here $IPS == PRNAP_{IPS} = IP(A)$ and $MACS = PRNAP_{MACS} = MAC(D)$).
  Similarly, Packet 4 is not malformed, sent to IDS but not present in $DADT$. So Algorithm 5.2 intimates event $PRNAP$ and M_d moves to state s_7 via measured

transition t_{M6}. In state s_6 of M_d enabling event of t_6 is $PRNAP$ with checking condition of model variables as $IPS = PRNAP_{IPS}$ and $MACS! = PRNAP_{MACS}$ (here $IPS = PRNAP_{IPS} = IP(A)$ and $MACS! = PRNAP_{MACS} = MAC(A)$).

Packet sequence 1 to 4 results in the following observed transition-trace $t_{M1}, t_{M2}, t_{M4}, t_{M6}$ which is not generated by T_2. $t_{M1}, t_{M2}, t_{M4}, t_{M6}$ corresponds to p_1, p_2, p_3, p_4, p_5 which violates $(\neg p_5 \wedge (p_3 \rightarrow X\neg p_6)) \cup p_6$. Thus, the detector outputs 'fault' for the trace and attack is detected. So, it may be noted that the probe $PRNSP$ leads to two NAPs one having mismatching $IP - MAC$ pair compared to the NSP being verified. So the change in sequence of NDP packets was a result of probing which in turn lead to detection of the attack. The $SPOOFT$ will be updated as follows:

Table 5.1: Spoofed Table: NS spoofing attack

IP_{Src}	MAC_{Src}	Ti
IP A	MAC D	$time\_of\_receipt$

Note: NA spoofing attack is very similar to NS spoofing attack, where instead of the NS packet carrying the spoofed $IP - MAC$ pair, NA packet is utilized by the attacker for spoofing. So, the same scheme with minor modifications can handle NA spoofing and it not elaborated in this chapter. In the next subsection we discuss the detector design for DAD attack.

5.4.4 Duplicate Address Detection Attack

In this sub-section we present the system modeling of NDP packets under DAD attack attempts (i.e., M_d, Section 5.3). Also, LTL specification of normal NDP behavior with its Buchi Automaton representation (i.e., B_f, Section 5.3) and models for DAD attack detector construction (i.e., $T_1, T_2, T_2', T_3, M_{T_2}$, Section 5.3) are presented.

5.4.4.1 DES Modeling of DAD attack

The various components of the DES model under normal/DAD attack scenario are as follows.

Model Variables

Model variables in the DES model of DAD are $IPT, MACS, MACS1, IPR, MACR$.
Domain of IPT, IPR is $\{d.d.d.d \mid d \geq 1 \; and \; d \leq 255\}$.

Domain of $MACS$, $MAC1$ and $MACR$ is $\{d.d.d.d.d.d \mid d \geq 1 \; and \; z \leq 255\}$.

Event Set

The following changes in the LAN are considered as events for a DAD:

- *NSPDAD*1 denotes the receipt of DAD *NSP* which has not been previously encountered.
- *NSPUDAD*2 denotes the receipt of DAD *NSP* whose target IP address is same as that *NSPDAD*1.
- *NAPDAD* denotes the receipt of DAD *NAP* which is a reply to *NSPDAD*1 packet.
- *DTD* occurs when the IP-MAC pairing of the DAD packet is already present in Authenticated/Spoofed table.
- *PRNSPUPDAD* denotes the sending of an unicast *NS* probe packet for verifying reachability of
  $NAPDAD_{IPS}$.
- *PRNAPUPDAD* denotes the receipt of the *NAP* as a probe response to *PRNSPUPDAD*.
- *PRNSPDAD* denotes the sending of NS probe packet to perform DAD for IP_{rand} (random IP from AUTHT table).
- *PRNAPDAD* denotes the receipt of the DAD *NAP* as a probe response (to *PRNSPDAD*) such that source IP of the DAD *NAP* matches with IP_{rand} and source MAC of DAD *NAP* matches with MAC_{rand}.
- *PRNSPRCVDAD* denotes the receipt of a DAD *NSP* as a response to *PRNSPDAD*.
- *EXP* the event EXPHAND() It triggers the *EXP* event.
- u is an unobservable event.

Transitions

Transitions follow the same three-valued tuple definition as given for *NS* spoofing. The transitions are illustrated in the DES model given in Figure 5.15.

Proposition Set

The following are the set of propositions:

- p_1 = *System is in its initial state.*
- p_2 = *NSP is obtained with* NSP_{IPS} *as unspecified and no previous entry in the DADT.*
- p_3 = *NAP is obtained with* NAP_{IPD} *as all node multicast address and* NAP_{IPS} *is matching with target address of* $NSPDAD1$*, i.e.,* $NAP_{IPS} = IPT$*.*
- p_4 = *NSP is obtained with* NSP_{IPS} *as unspecified and target IP of NSP is same as target IP of* $NSPDAD1$*, i.e.,* $NSP_{IPS} = IPT$
- p_5 = *unicast NS probe packet is sent to check if host having* $NAPDAD_{IPS} - NAPDAD_{MACS}$ *is up i.e., check if* $IPT - MACS1$ *is up.*
- p_6 = *NA probe response is received from* $NAPDAD_{IPS} - NAPDAD_{MACS}$*, i.e.,* $IPT - MACS1$ *is up.*
- p_7 = *NS probe is sent with source IP as ::, target IP address as some random IP* IP_{rand} *from AUTHT and source MAC as* $NSPDAD1_{MACS}$
  i.e., probe claims that host with MACS wants to configure IPR.
- p_8 = *NAP probe response is received with* NAP_{IPS} *matching the target IP in PRNSPDAD and* NAP_{MACS} *matching source MAC of PRNSPDAD i.e., NA probe response from host having IP as IPR and MAC as MACR.*
- p_9=*NAP probe response is received with* NAP_{IPS} *matching the target IP in PRNSPDAD and* NAP_{MACS} *not matching source MAC of PRNSPDAD*
  i.e., NA probe response from host having IP as IPR and MAC different from MACR.
- p_{10}=*NS probe response is received with* NSP_{IPS} *as :: and* NSP_{IPT} *matching with target IP in PRNSPDAD*
  i.e., NS probe response from attacker host stating to configure IPR.

- p_{11} = (T_{req} expires) OR
  (IP/MAC pairing under question is already present in $AUTHT/SPOOFT$ table).

Mask Function

The mask function M used is defined as follows:
$M(NSPDAD1) = NSPDAD1$, $(NSPDAD2) = NSPDAD2$, $M(NAPDAD) = NAPDAD$, $M(DTD) = DTD$,
$M(PRNSPUPDAD) = PRNSPUPDAD$,
$M(PRNAPUPDAD) = PRNAPUPDAD$,
$M(PRNSPDAD) = PRNSPDAD, M(PRNAPDAD) = PRNAPDAD, M(PRNSPRCVDAD) = PRNSPRCVDAD$,
$M(EXP) = EXP, M(u) = \epsilon$.

So, the DES model for DAD under normal and attack condition, shown in Figure 5.15, can be represented is follows:

$$M_d = (S_i, S_{i0}, V, \Sigma, \tau, R, AP, L) \text{ where,}$$

- $S_i = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}\}$.
- $S_{i0} = \{s_1\}$.
- $V = \{IPT, MACS, MACS1, IPR, MACR\}$.
- $\Sigma = \{NSPDAD1, NSPDAD2, NAPDAD, PRNSPUPDAD, PRNAPUPDAD, PRNSPDAD, PRNAPDAD, PRNSPRCVDAD, EXP, DTD, u\}$ is the set of events.
- $\tau = \{t_1, \cdots t_{21}\}$
- R is the transition relation shown in Figure 5.15.
- $AP = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}\}$.
- L is the labeling function shown in Figure 5.15.

An overview of the DES model for the normal/ attack scenario is now explained. Under normal condition, as shown in Figure 5.15, event $NSPDAD1$ triggers transition t_1

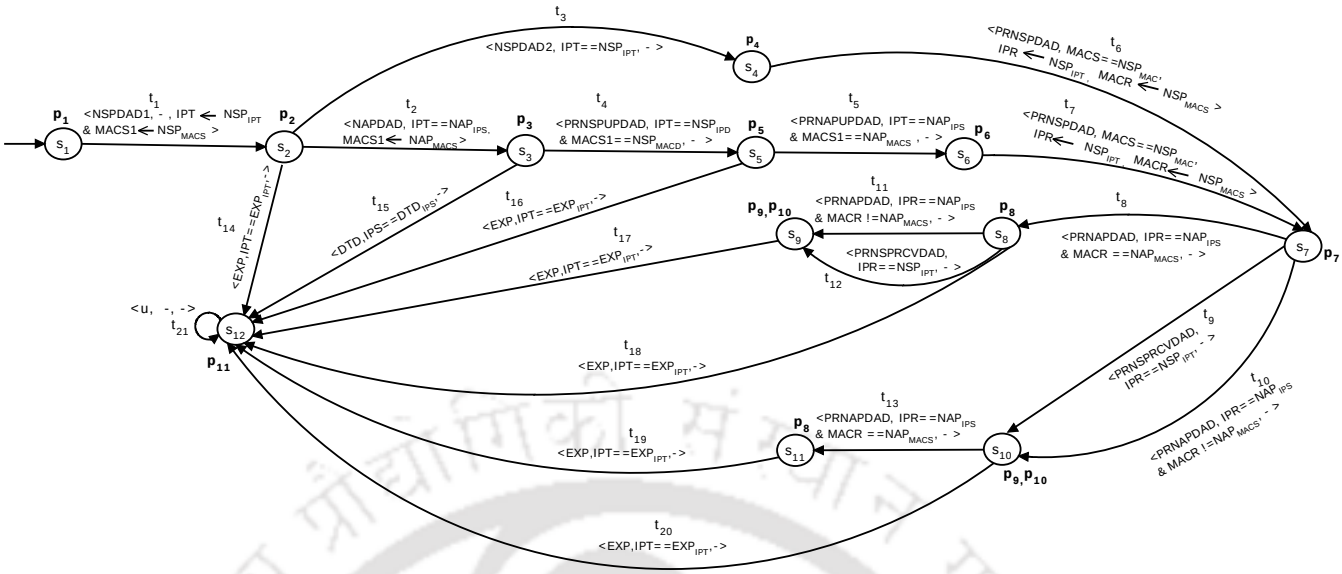

Figure 5.15: DES model for DAD attack

from state s_1 to s_2. The model variables *IPT* and *MACS* are assigned with target IP address and source MAC address of the *NSP* which triggers *NSPDAD*1. At state s_2, proposition p_2 holds. If no advertisement or solicitation packet arrives from any host claiming *IPT*, before *EXP* is triggered, it implies that the targeted IP (i.e., *IPT*) is not in use and the entering host can take it. *EXP* causes the transition t_{14} from s_2 to s_{12}, where p_{11} holds. Else, if an *NAP* arrives with source IP address same as that of *IPT* (i.e., event *NAPDAD*), state s_3 is reached via transition t_2 where p_3 holds. *MACS*1 is assigned with the source MAC address of the *NAP* packet. If an *NSP* arrives whose target IP is same as that of *IPT* it is a response to *NSPDAD*1 and state s_4 is reached via transition t_3 (i.e., event *NSPDAD*2) where p_4 holds. If the source $IP - MAC$ pair of *NAPUN* packet is already present in the *AUTHT*, *DTD* event triggers transition t_{15} that takes the system to s_{12}. Otherwise, an unicast *NS* probe request is sent to $IPT - MACS1$ to verify its reachability and event *PRNSPUPDAD* causes the transition t_3 to state s_5. Under normal condition, one advertisement in reply to the *NS* probe packet arrives within t_{req} time and the system moves to state s_6 via transition t_5 where p_6 holds. The reply is identified by matching the NAP_{IPS} with *IPT* and NAP_{MACS} with *MACS*1. If the host is reachable, either the host is genuine or it is the attacker. So another phase of active probing is performed. A random IP, IP_{rand} (whose MAC is MAC_{rand}) is selected from the *AUTHT* table. A solicitation DAD probe is sent as if $NSPDAD1_{MACS}$ is performing a *DAD* for an alternative (random) IP. More specifically, an *NS* probe is sent by IDS having

target IP as IP_{rand} and source MAC as $NSPDAD1_{MACS}$, which acts as if the host (who sent $NSPDAD1$) is trying to select IP_{rand} as its alternative new IP address; $PRNSPDAD$ causes the transition t_7 to state s_7 where p_7 holds. A similar probe is also sent after $NSPDAD2$ arrives; t_6 corresponds to the probe. In the probes IP_{rand} and MAC_{rand} are stored in variables IPR and $MACR$, respectively and condition being checked is $MACS == NSP_{MACS}$. Under normal conditions, the genuine host will reply to this $PRNSPDAD$ with an NAP whose source IP and MAC matches with $IP_{rand} - MAC_{rand}$. The system moves to s_8 via t_8 where it waits for EXP which finally takes the system to s_{11}, where the system terminates. To make the system infinite a self loop t_{21} i.e."$u, -, -$" (an unobservable event) has been added to s_{12}.
Now we discuss the model under attack condition. Most of the transitions are similar to those of the normal condition and here we will explain only those which are different. After the probe $PRNSPUPDAD$ (used to check reachability of $IPT - MACS1$), under attack condition no reply arrives from host having $MACS1$ until t_{req} time of sending the probe; $MACS1$ was a spoofed address used by attacker in $NAPDAD$. Transition t_{16} takes the system from s_5 to state s_{12}, enabled by event EXP. Also, after the probe $PRNSPDAD$ (state s_7), under attack condition two events are possible–(i) NS packet arrives claiming IP_{rand} as target IP and event $PRNSPUNDAD$ causes transition t_9 to state s_{10}; (ii) NA probe response from attacker with source $IP - MAC$ pair different from that of $IP_{rand} - MACrand$ arrives and transition t_{10} occurs moving the model to state s_{10}. It may be noted that as IP_{rand} was selected from $AUTHT$, the corresponding host is up and it should reply with an NAP having IP-MAC as $IP_{rand} - MAC_{rand}$. Also, there are very little probability that a genuine host enters the network and claims IP_{rand} as target at the same time when the probe is sent. Similar two options are possible at s_8. Even if the system is under attack, an NA probe response from normal host (source $IP - MAC$ pair same as that of $IP_{rand} - MACrand$ arrives and transition t_{13} occurs moving the model to state s_{11}. Once all the responses for $PRNSPDAD$ arrive the model moves to state s_{12} i.e., t_{18} and t_{19}.
The LTL specification designating the non faulty behavior (normal scenario when there is no attack) of the detector in terms of the events is given as:
Either no $NAPDAD$ arrives and the system encounters EXP right after $NSPDAD$ OR DTD event occurs after receipt of $NAPDAD$ packet OR probe $PRNSPDAD$ receives a single genuine advertisement $PRNAPDAD$ (such that its source $IP - MAC$ is same as that of the random $IP - MAC$ pair used in the probe) as response after which EXP occurs.

So in terms of the transition set, the specification can be given as:
Either no t_2 (caused by *NAPDAD*) occurs and the system encounters t_{14} (caused by *EXP*) right after t_1 OR t_{16} occurs after t_2 OR probe t_7/t_6 receives a single genuine advertisement as response causing transition t_8 after which t_{18} occurs.
Thus, the above specification is given by the LTL formula f_1 as:
$F(p_2 \wedge Xp_{11}) \vee F(p_3 \wedge Xp_{11}) \vee F(p_7 \wedge Xp_8 \wedge XXp_{11})$

Proof of correctness of this specification can be shown similarly as shown in case *NS* spoofing attack. For brevity, the description is omitted here.

5.4.4.2 Testing diagnosability and building detector

Now the detector of the IDS is synthesized following same steps as stated in Section 5.3.

1. **Formation of Buchi Automata**
   As stated earlier first a Buchi automata for the specification f_1 is derived as:

$$B_{f_1} = \left(C_{f_1}, \Sigma_{AP}, R_{f_1}, q_0^{f_1}, Fi\right)$$

   which is shown in Figure 5.16 where,
   $C_{f1} = \{B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10}, B_{11},$
   $B_{12}, B_{13}, B_{14}, B_{15},\}$ is the set of states.
   $\Sigma_{AP} = \{p_1, p_2, p_3, p_4, p_5, p_6, p_8, p_9, p_{10}, p_{11}\}$ is the event set.
   R_f is the transition relation as shown in Figure 5.16.
   $q_0^{f_1} = B_0$ is the initial state.
   $Fi = \{B_{14}, B_{15}\}$ are the final F_1 labeled states.
   The specification can be said to be a conjunction of three parts. We will show how a part of the specification (i.e., $F(p_7 \wedge Xp_8 \wedge XXp_{11})$) is satisfied by the Buchi automata; others can be interpreted similarly.

   It can be seen from the Figure 5.16 that B_f has six paths emanating from the initial state B_0. The rightmost path $\langle B_0, B_{13}, B_{12}, B_{14}, \cdots \rangle$ satisfies $p_7 \wedge Xp_8 \wedge XXp_{11}$ in B_{13} p_7 holds, in the next state i.e., B_{13} p_7 holds and in the next to next state i.e., B_{14} p_{11} holds. In a similar manner, in path $\langle B_0, B_9, B_{10}, B_{11}, B_{12}, B_{14}, \cdots \rangle$ also, the part of the

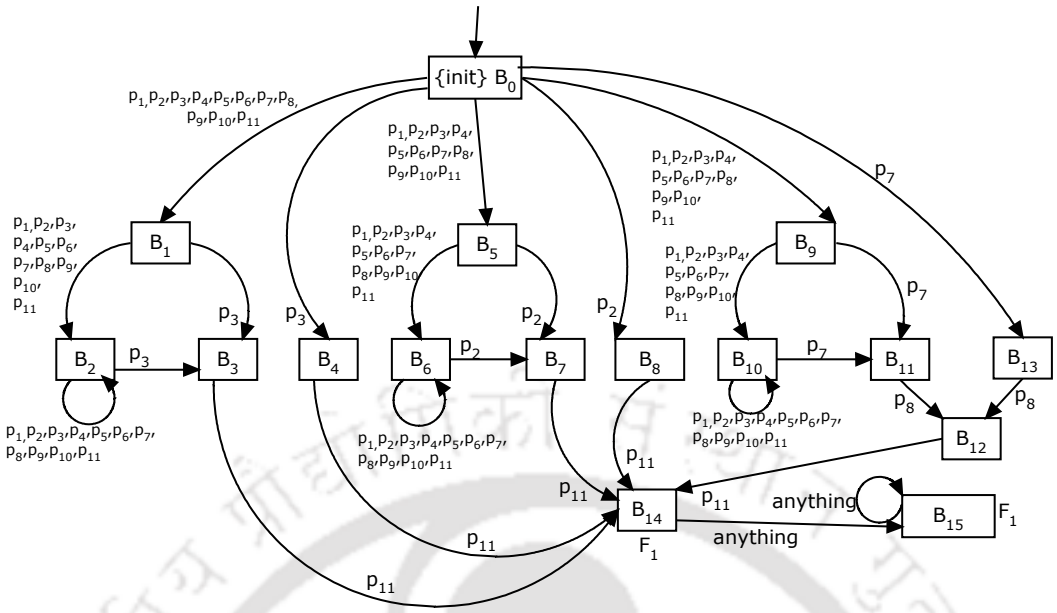

Figure 5.16: Buchi Automata T_{f1} for f_1

specification holds.

It can be verified similarly that B_f accepts all proposition traces over AP satisfying f_1.

2. **Proposition Synchronization of Buchi automata and the Model for test of pre-diagnosability**

   Now proposition synchronization T_1 of M_d and B_f is constructed. It is shown in Figure 5.17.

$$T_1 = (Q_1, \Sigma, \tau, R_1, Q_0^1, AP \cup Fi, L_1) \text{ where,}$$

- $Q_1 = \{y_{11}, y_{22}, y_{33}, y_{10\,3}, y_{10\,4}, y_{10\,5}, y_{10\,6}, y_{11\,7}, y_{12\,8}, y_{14\,12}, y_{15\,12}\}$
- Σ = set of events same as that of the model M_d.
- $\tau\{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_{14}, t_{15}, t_{18}, t_{21}\}$.
- R_1 is the transition relation shown in Figure 5.17.
- $Q_0^1 = \{y_{11}\}$.
- $AP \cup Fi$ is the new set of proposition.
- L_1 is the labeling function shown in Figure 5.17.

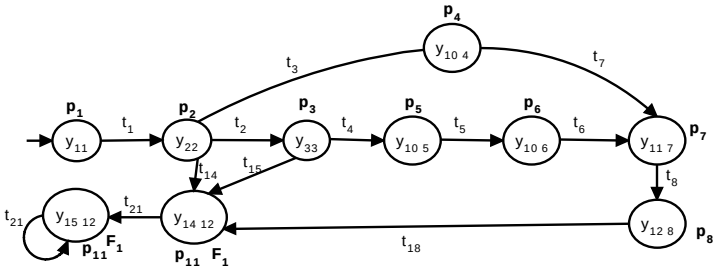

Figure 5.17: Model T_1 for f_1

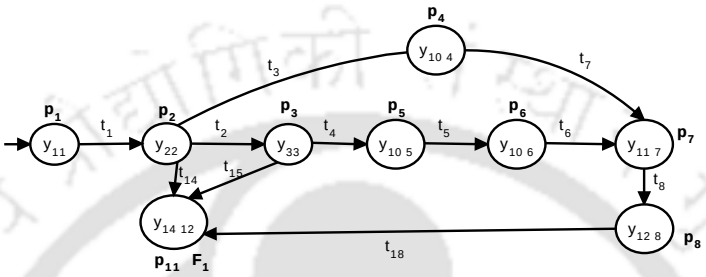

Figure 5.18: Model T_2 for f_1

It can be clearly concluded from Figure 5.17, that there is no path in T_1 that does not visit $y_{15\,12}$ infinitely often and $y_{15\,12}$ is one of the states satisfying F_1. Thus it can be stated that T_1 satisfies LTL specification $G\mathbb{F}F_1$.

3. **Testing Diagnosability**

   As the system model passes the pre-diagnosability test, T_2 is constructed for diagnosability test.

   Synthesis of Masked T_1*:* Using the mask function defined earlier, T_2 is constructed as shown in Figure 5.18 and defined as follows:

$$T_2 = (Q_2, \Delta, \tau_M, R_2, Q_0^2) \text{ where}$$

- $Q_2 = \{y_{11}, y_{22}, y_{33}, y_{10\,3}, y_{10\,4}, y_{10\,5}, y_{10\,6}, y_{11\,7}, y_{12\,8}, y_{14\,12}\}$
- $\Delta = \{NSPDAD1, NSPDAD2, NAPDAD, PRNSPUPDAD, PRNAPUPDAD, PRNSPDAD, PRNAPDAD, PRNSPRCVDAD, EXP, DTD\}$.
- $\tau_M = \{t_{M1}, t_{M2}, t_{M3}, t_{M4}, t_{M5}, t_{M6}, t_{M7}, t_{M8}, t_{M14}, t_{M15}, t_{M18}\}$.

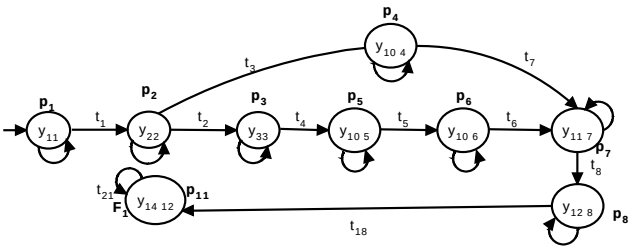

Figure 5.19: Model T_2' for f_1

- R_2 is the transition relation as shown in Figure 5.18.
- $Q_0^2 = \{y_{11}\}$.

Now from T_2, T_2' is constructed as shown in Figure 5.19.

$$T_2' = (Q_2, \Sigma, \tau, R_2', Q_0^2) \text{ where}$$

- $Q_2 = \{y_{11}, y_{22}, y_{33}, y_{10\,3}, y_{10\,4}, y_{10\,5}, y_{10\,6}, y_{11\,7}, y_{12\,8}, y_{14\,12}, y_{15\,12}\}$
- $\Sigma = \{NSPUN1, NSPUN2, NAPUN, DTD, PRNSPUP, PRNAPUP, PRNSPDAD, PRNAPDAD1, PRNAPDAD2, PRNSPUNDAD, EXP, u\}$.
- $\tau = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_{14}, t_{15}, t_{18}, t_{21}\}$.
- R_2' is shown in Figure 5.19.
- $Q_0^2 = \{y_{11}\}$.

Diagnosability Tested by Model Checking:
To test the diagnosability, T_3 is constructed from the event synchronization of T_2' and the system model as shown in Figure 5.20.

$$T_3 = (Q_3, \Sigma, \Im, R_3, Q_0^3, AP, L_3) \text{ where,}$$

- $Q_3 = \{y_{11}, y_{22}, y_{33}, y_{10\,3}, y_{10\,4}, y_{10\,5}, y_{10\,6}, y_{11\,7}, y_{12\,8}, y_{14\,12}, y_{15\,12}\}$

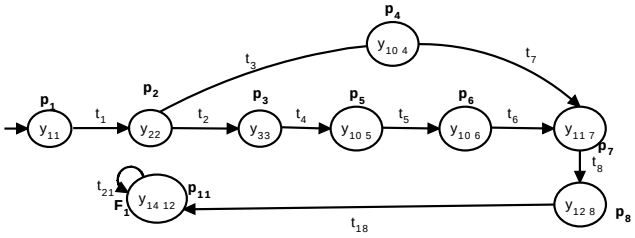

Figure 5.20: Model T_3 for f_1

- Σ =, same as M_d
- $\Im = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_{14}, t_{15}, t_{18}, t_{21}\}$.
- R_3 is transition relation shown in Figure 5.20.
- $Q_0^3 = \{y_{11}\}$.
- AP is the proposition set, same as M_d.
- $L_3 = Q_3 \rightarrow 2^{AP}$ is the labeling function, shown in Figure 5.20.

As already discussed, if M_d is diagnosable then every infinite proposition trace generated by T_3 must satisfy f_1. Again it is an LTL model checking problem that was solved using NuSMV model checker; the system was found to be diagnosable with respect to f_1.

4. **Final Detector**

   As model is diagnosable with respect to the specification, its detector can be constructed. As already discussed, T_2 along-with M_{T_2} forms the final detector.

   In case of DAD attack, if $t_1, t_3, t_6, t_8, t_{18}$ is a transition trace generated by M_d, its observation equivalent transition trace is $t_{M1}, t_{M3}, t_{M6}, t_{M8}, t_{M18}$ which is generated by T_2. $t_{M1}, t_{M3}, t_{M6}, t_{M8}, t_{M18}$ corresponds to $p_1, p_2, p_4, p_7, p_8, p_{11}$ which adheres to $F(p_7 \wedge Xp_8 \wedge XXp_{11})$ (which is a part of the specification). As the specification is conjunction of three parts and the trace satisfies a part of it, the whole specification is satisfied. Thus, the detector outputs 'normal' for this case. On the other hand, for t_1, t_2, t_4, t_{16} whose observation equivalent transition trace is $t_{M1}, t_{M2}, t_{M4}, t_{M16}$ is not generated by T_2. $t_{M1}, t_{M2}, t_{M4}, t_{M16}$ corresponds to $p_1, p_2, p_3, p_5, p_{11}$ which does not adhere to the specification. Thus, the detector outputs 'fault' for this case.

5.4.4.3 An Example to illustrate detection of *DAD* attack

For illustrating DAD attack using an example, the basic architecture used for *NS* spoofing (Figure 5.4 (subsection 5.4.1)) will be considered.

It is assumed that there is one entry in *AUTHT* and *DADT*, as shown in Table 5.2 and Table 5.3.

Table 5.2: Initial *AUTHT*: Illustration of DAD attack

IP_{Src}	MAC_{Src}	Ti
IP A	MAC A	*time_of_receipt*

Table 5.3: Initial *DADT*: Illustration of DAD attack

IPT	*MACS*	time of receipt	time of sending
IP A	MAC A	*time_of_receipt*	-

The following is the sequence of events:

- Packet 1: Host B comes up in the network and sends a DAD *NSP* (*NSPDAD*1) with target IP as $IP(B)$, source IP as :: and source MAC as $MAC(B)$.
  NS_HANDLER() (Algorithm 5.1) receives Packet 1. The packet is not malformed, not sent by the IDS, source IP address is :: and $IP(B)$ is not present in *DADT*. So, Algorithm 5.1 intimates event *NSPDAD*1 and M_d (Figure 5.15) moves to state s_2 via measured transition t_{M1}.
  Also $\langle IP(B), MAC(B), time\_of\_receipt, - \rangle$ is entered in *DADT* (shown in Table 5.4).

- Packet 2: Attacker D sends a DAD *NSP* (*NSPDAD*2) with target IP as $IP(B)$, source IP as :: and source MAC as $MAC(k)$ ($MAC(k)$ may be MAC of attacker or any arbitrary one).
  NS_HANDLER() (Algorithm 5.1) receives Packet 2. The packet is not malformed, not sent by the IDS, source IP address is ::, $IP(B)$ is present in *DADT* and time of receipt of packet 2 is within t_{req} of receipt of packet 1. So, Algorithm 5.1 intimates event *NSPDAD*2 and M_d (Figure 5.15) moves to state s_4 via measured transition t_{M3}.

- Packet 3: Algorithm 5.1 selects a random IP IP_{rand} from *AUTHT* (here, $IP_{rand} = IP(A)$) and sends a DAD probe with target IP as $IP(A)$, source MAC as $MAC(A)$ and source

IP as ::. Event *PRNSPDAD* is generated. Also $\langle IP(A), MAC(A), -, time\_of\_sending \rangle$ is entered in *DADT* (shown in Table 5.4). Finally, *EXPHAND* is called with $IP(A)$ as parameter to check passage of t_{req} time after sending the probe. M_d (Figure 5.15) moves to state s_7 via measured transition t_{M6}.

- Packet 4: Within t_{req} time of sending *PRNSPDAD*, the genuine host (i.e., having $IP(A)$) will respond with a DAD *NAP* (i.e., *NAP* with solicitation flag set) where source IP-MAC is $IP(A) - MAC(A)$ and destination IP is all node multicast. As $IP(A)$ is present in *DADT*, NA_HANDLER() (Algorithm 5.2) intimates event *PRNAPDAD* and M_d moves to state s_8 via measured transition t_{M8}. In state s_7 of M_d (Figure 5.15) enabling event of t_8 is *PRNAPDAD* with checking condition of model variables as $IPR == NAP_{IPS}$ and $MACR = NAP_{MACS}$ (here $IPR == NAP_{IPS} = IP(A)$ and $MACR = NAP_{MACS} = MAC(A)$).

- Packet 5: Within t_{req} time of sending *PRNSPDAD*, the attacker will respond with a DAD *NAP* (as it is assumed that attacker wants to keep DAD attack persistent), where source IP-MAC is $IP(A) - MAC(k)$ and destination IP is all node multicast. NA_HANDLER() (Algorithm 5.2) intimates event *PRNAPDAD* and M_d moves to state s_9 via measured transition t_{M11}. In state s_9 of M_d (Figure 5.15) enabling event of t_{11} is *PRNAPDAD* with checking condition of model variables as $IPR == NAP_{IPS}$ and $MACR! = NAP_{MACS}$.

Packet sequence 1 to 5 results in the following observed transition-trace $t_{M1}, t_{M3}, t_{M6}, t_{M8}, t_{M11}$ which is not generated by T_2. $t_{M1}, t_{M3}, t_{M6}, t_{M8}, t_{M11}$ corresponds to $p_1, p_2, p_4, p_7, p_8, p_9, p_{10}$ which violates the LTL formula.

Thus, the detector outputs 'fault' for the trace and attack is detected.

Table 5.4: Updated *DADT*: Illustration of DAD attack

No.	*IPT*	*MACS*	time of receipt	time of sending
0.	IP A	MAC A	$timeofreceipt - 0$	-
1.	IP B	MAC B	$timeofreceipt - 1$	-
2.	IP B	MAC k	$timeofreceipt - 2$	-
3.	IP A	MAC A	-	$timeofsending - 3$

The *SPOOFT* will be updated as follows:

Table 5.5: Spoofed Table: *DAD* spoofing attack

IP_{Src}	MAC_{Src}	Ti
IP B	MAC k	$timeofreceipt$

5.5 Experimentation and Result

For benchmarking LTL based IDS for NDP attacks, we considered metrics of accuracy, detection rate, resource consumption etc. As some extra NDP messages are generated for probing, we also considered extra traffic statistics also as a benchmark along with accuracy, detection rate and resource consumption. The results in different situation of the network traffic is given as follows.

As the domain of most of the NDP messages are within same subnet, to the best of our knowledge, no standard dataset is available to test our proposed scheme. So, a test bed has been created for verifying the proposed scheme. As only limited tools were available for attacking IPv6 Network, we used the available suite *thc-ipv6* and build our own attack suit for most of the cases. The testbed consists of 6 machines running different operating systems. We name the machines with alphabets ranging from A-F. Machines A-E are running the following OSs: Windows Xp, Fedora 16 (as router), Windows 7, Backtrack 5, Ubuntu and Windows 2008, respectively. The machine D with Backtrack 5 is acting as the attacker machine and machine E is set up as the IDS. These machines are connected in a LAN with a CISCO catalyst 3560 G series switch [57] with port mirroring enabled for system E. The algorithms are implemented in C++. The IDS has two preemptive modules namely, packet grabber and packet injector. Packet grabber sniffs the packets from the network, filters NDP packets and invoke different modules.

5.5.1 Accuracy, Detection Rate and Traffic Overhead

For analyzing accuracy and detection rate, a study is also made to check tradeoffs regarding detection rate versus system performance. In other words, we have performed five experiments, where randomly 100%, 80%, 60%, 40% and 20% of NDP packets were selected to be verified with probing and detection rate, accuracy was verified.

5.5.1.1 Detection rate of NS/NA Spoofing Attack

In case of NS/NA spoofing, accuracy is always 100% (i.e., false positives are 0) because NS/NA spoofing is detected when i) the IP-MAC pair under question is found in *SPOOFT* i.e., trace t_1, t_{12} in Figure 5.5 or ii) for a probe, a response arrives with non-matching MAC address i.e., trace t_1, t_2, t_3 or t_1, t_2, t_4, t_6 in Figure 5.5 or iii) for a probe no response arrives i.e., trace t_1, t_2, t_7 in Figure 5.5. Case i) comprises already detected spoofed IP-MAC pairs and Case ii) and Case iii) can never happen under normal condition. So, the scheme never reports a normal case as attack, resulting in accuracy of 100%. It may be noted detection rate is not 100%, i.e., some false negatives are present. This is because when attacker spoofs itself case (ii) can be prevented; this is explained as follows. If an attacker (with IP-MAC address as $IP(A) - MAC(A)$, say) sends an NS/NA packet with source IP-MAC as IP(A)-MAC(C), then the situation is "attacker spoofs itself. IDS sends a probe to IP(A). Only one crafted *NAP* from attacker as $IP(A) - MAC(C)$ will arrive following the probe sent by IDS; for the probe, a response arrives with *matching* MAC address. Thus IDS incorrectly detects the case as a genuine one. Among the different attack NA/NS spoofing scenarios generated, 10% were the ones where the attacker spoofs itself. This number was intensionally kept low, because the scenario where attacker spoofs itself does not have any impact. For an attacker associating its own IP with a different MAC will redirect packets that are destined to it to the other host, whose MAC is associated with its IP.

Table 5.6, 5.7 illustrates the detection rate versus percentage of packets being probed. It is established that with tradeoff in the number of probe packet set, the detection rate does not fall much. With merely 20% of probe packets, the detection rate is still above 50%. This is because, over a period of time we build the *AUTHT* where all the genuine IP-MAC pairs are recorded and *LOGT* where spoofed IP-MAC pairs are recorded thereby can detect attack by looking at these tables only and doesn't require sending of probes. Also the number of false positives is always zero in all the cases and the false negatives is robust enough with tradeoff in probe packets.

5.5.1.2 Traffic Overhead of NS/NA attack

Figure 5.21, Figure 5.22shows the amount of extra NDP traffic generated due to the verification probe sent for verifying NS/NA messages. The plot in 5.21, shows the amount

Table 5.6: NS Attack statistics

% of probe packet	no. of attacks launched	Detection rate
100	116459	100.00
80	116459	90.95
60	116459	79.48
40	116459	67.08
20	116459	54.69

Table 5.7: NA Attack statistics

% of probe packet	no. of attacks launched	Detection rate
100	160951	100.00
80	160951	92.43
60	160951	80.10
40	160951	66.78
20	160951	54.22

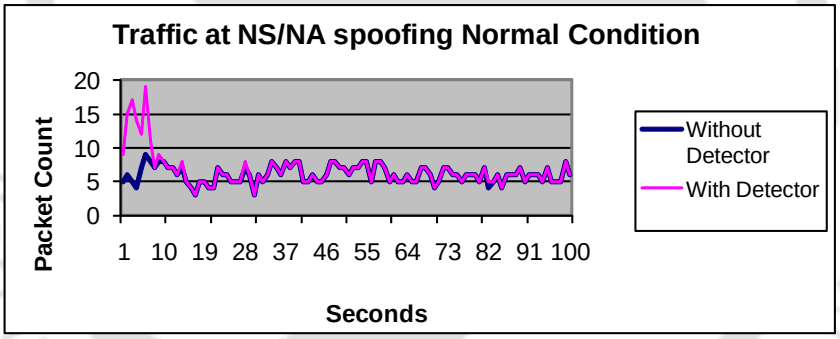

Figure 5.21: NDP traffic at NS/NA Normal Condition

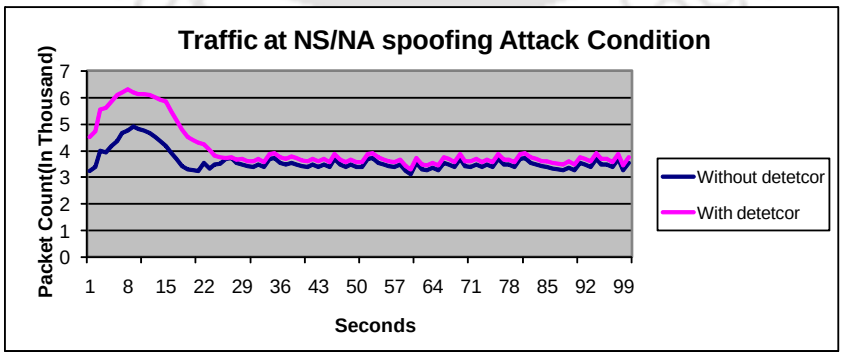

Figure 5.22: NDP traffic at NS/NA Spoofed Condition

of traffic in normal operation and in the presence and absence of the IDS. We notice that the extra traffic generated is slightly high at the initialization phase and then goes down to become negligible. Once genuine IP-MAC pairs are identified (by probing) they are stored in Authenticated bindings table. Following that no probes are required to be sent for any NDP solicitation/advertisement from these IP-MAC pairs.

Plot in Figure 5.22 shows the traffic generated when IDS is running and there are spoofing attacks in the network. In this case, a little extra traffic is generated by our IDS for the probes. With each spoofed NS/NA packet, our IDS sends a probe request and expects at most two replies (one from normal and the other from the attacker), thereby adding only three NDP packets for each spoofed packet. This extra traffic also becomes considerably small as the genuine IP-MAC pairs are identified and stored in *AUTHT* and spoofed IP-MAC pair is stored in *LOGT*. Looking at this two tables, most of the spoofing for existing IP/MAC can be identified without sending a probe after the initialization period.

5.5.1.3 Detection rate of DAD Attack

For DAD attacks the accuracy is 100% because the attack is detected when (i) the IP-MAC pair under question is found in *SPOOFT* i.e., trace t_1, t_2, t_{12} in Figure 5.15, (ii) DAD probe targeting an existing (randomly selected from *AUTHT*) IP address is sent and the response has a non-matching MAC address i.e., traces
$t_1, t_2, t_4, t_5, t_7, t_{10}$ or $t_1, t_3, t_6, t_7, t_{10}$ in Figure 5.15, (iii) DAD probe targeting an existing IP address is sent and the response is a DAD *NSP* also targeting the same IP address i.e., traces t_1, t_3, t_6, t_7, t_9 or $t_1, t_2, t_4, t_5, t_7, t_9$ in Figure 5.15, (iv) no response arrives for a probe sent to check if a host (which has sent a DAD *NAP* in response to DAD *NSP*) is up i.e., trace t_1, t_2, t_4, t_{16} in Figure 5.15. These three cases can never happen under normal condition thereby resulting in accuracy of 100%. The detection rate is however lower than 100% (i.e., some attacks could not be detected) explained as follows. Among the four cases mentioned above Case (i) detects the attack from history and hence cannot be prevented by attacker. Case (ii) and Case (iii) can be prevented by attacker if it can somehow guess when a DAD *NSP* is a probe from IDS and when a DAD *NSP* arrives from an entering host; if the DAD *NSP* is a probe from IDS the attacker do not respond. This results in failing to detect the DAD attack. It may be noted that as random IP address are used in the DAD probes it is

difficult for a attacker to guess with good accuracy. Further, a lot of false guesses may lead to entering hosts obtain IP address easily, leading to ineffectiveness of DoS caused by DAD attack. In our experiments the attacker was set not to reply to few DAD *NSP*s (selected randomly). Case (iv) can be prevented by attacker by replying to the probe sent to check if the host under question is up. However, once such a reply comes the IDS sends a DAD probe which leads to Case (ii) or Case(iii).

Table 5.8: DAD Attack statistics

% of probe packet	no. of attacks launched	Detection rate
100	160951	100
80	160951	79.77
60	160951	59.54
40	160951	39.31
20	160951	19.75

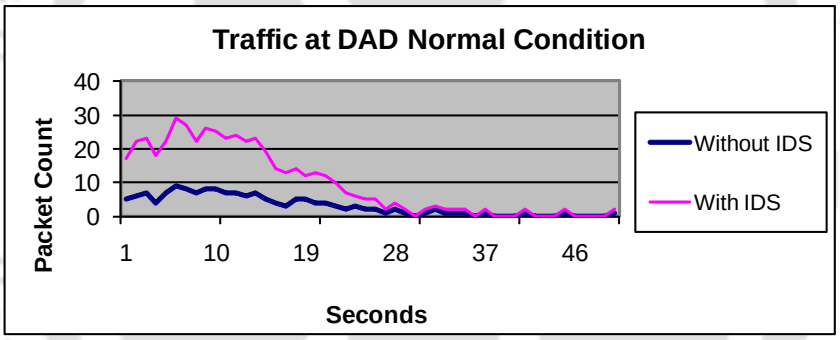

Figure 5.23: Traffic at DAD Normal Condition

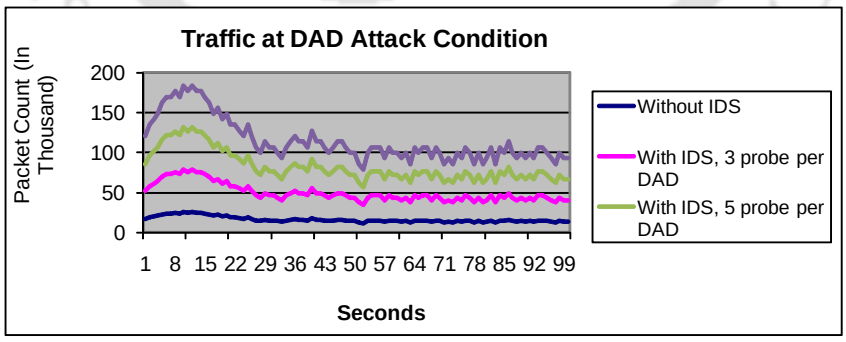

Figure 5.24: Traffic at DAD Attack Condition

Table 5.8 shows detection rate versus percentage of packets being probed with random number of DAD probe packets. With tradeoff in the number of probe packet sent, the

detection rate is linearly proportional. At 100% probe packet, the detection rate is exactly 100%. At 20% of probe packets, the detection rate falls to around 19%. Number of false positives is very close to zero in all the cases and the false negatives is proportional to the tradeoff in probe packets.

Figure 5.23, Figure 5.24shows the amount of extra NDP traffic generated due to the verification probe sent for verifying DAD messages. The plot shows the amount of extra traffic generated with 3,5 and 10 nos of probes sent for verifying DAD messages.

The plot in 5.23, shows the amount of traffic in normal operation and in the presence and absence of the IDS. It is noticed that the extra traffic generated is a little higher. This can be regulated by the number of probe messages sent against each DAD message.

Plot in Figure 5.24 shows the traffic generated when IDS is running and there are DAD attacks in the network. In this case, also similar amount of extra traffic is generated by the IDS for the probes. Similar to normal case , this can also be regulated by the number of probe messages sent against each DAD message.

5.5.2 Performance and Scalability

In the testbed, we injected different amount of attack packets (up to 2000) per second and measured CPU utilization of the processor running the IDS, memory utilization of the system running the IDS and bandwidth utilization in the LAN.

So, memory utilization, CPU utilization and bandwidth utilization in the LAN have been computed for different amount of probing. Figure 5.25, 5.26, 5.27 shows memory, bandwidth and CPU utilization when running our IDS in Intel Core-2-Duo machine (With Ubuntu 9.10).

Considering the NDP cache timeout in modern OS varies from 15 sec to a few minutes, the number of NDP packets in a LAN with about 100 hosts should not exceed a few hundred. Hence we can conclude that under very high number of attack packets also the IDS works without any problem in terms of resources.

It may be noted that in the worst case of our experiment (i.e., maximum number of attack packets injected), CPU utilization is around 4% (including consumption by OS), memory utilization is about 1.4 GB and bandwidth utilization is around 1Mbps. So, we can conclude that under high number of attack packets also the IDS works without any

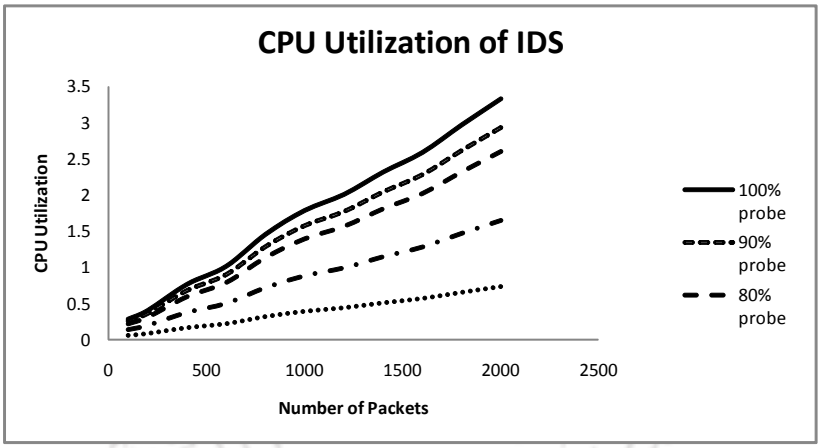

Figure 5.25: CPU Utilization of IDS

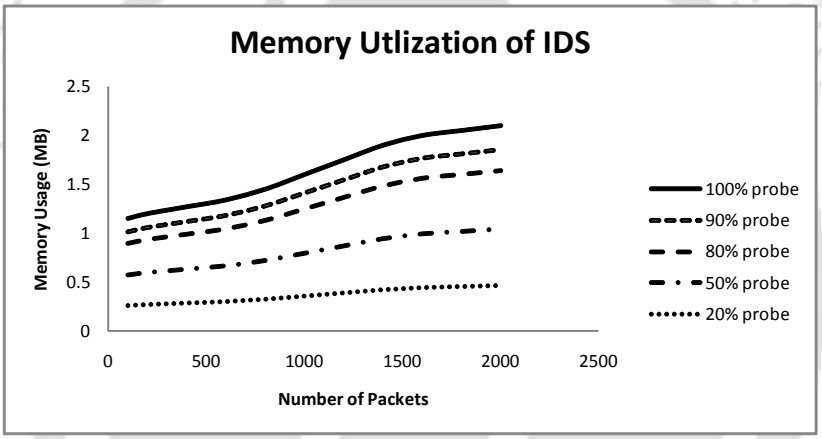

Figure 5.26: Memory Utilization of IDS

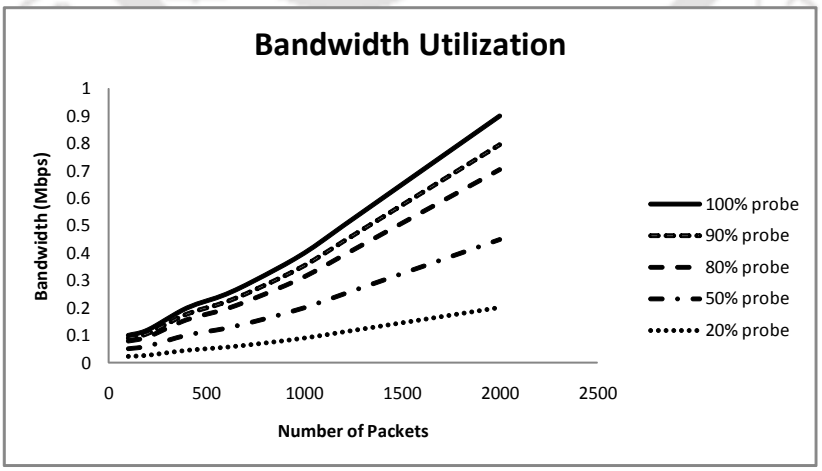

Figure 5.27: Bandwidth Utilization of IDS

problem in terms of resources.

5.6 Conclusion

In this chapter, an LTL based DES framework [43] has been proposed for building an active IDS to detect NDP related attacks. It is first discussed that passive IDSs like signature and anomaly systems cannot detect NDP related attacks efficiently. Following that the advantages of using LTL framework in specifying the NDP specification (under active probing) from natural language, automatic building process of the IDS and checking its correctness are highlighted. This LTL framework (of [43]) was augmented with model variables to avoid state explosion problem incurred in modeling the NDP. It is shown that adding model variables does not increase the complexity of the IDS construction procedure and checking its correctness. The IDS does not violate the standard NDP because the only additional requirement are probes, which are nothing but NDP related packets. IDS is launched on a single host inside the network; so, the scheme does not require patching every host in the network. Further, hardware requirement is just a switch with port mirroring.

Experimental results illustrate 100% detection rate and accuracy of 100% for NDP attacks when all NS/NA and DAD packets are probed. Detection rate falls if we decrease the probing rate. However, even for 100% probing, the maximum resource overhead in terms of CPU utilization, memory utilization and bandwidth utilization were minimal compared to modern network and system capacities.

Chapter 6

Conclusions and future scope of work

With the exponential growth in the number of connected hosts in the Internet, security threat of the hosts are ever increasing. Although there are many techniques like authentication, authorization, encryption, access control, firewall etc. to deal with these threats, none of them are sufficient to mitigate their persistence. An Intrusion Detection System(IDS) on the otherhand is a device or software application that determines the presence of malicious activities and produces alarm before the effect of the threat becomes widespread. The field of IDS has grown and evolved significantly in recent years. However, there are certain classes of known attacks for which the two main types of IDSs i.e., signature based IDS and anomaly based IDS can not generate effective alarms. For these attacks, signatures cannot be written because they do not change the syntax and sequence of network packets. Moreover, as there is no significant statistical behavior change because of these attacks, anomaly IDS generates a large number of false alarms. Since the network is designed in layered approach and nature of attack at each layer is different from the other, the IDSs also need to be designed in a layerwise fashion.

Recently, with phenomenal growth in man made and highly complex dynamic systems such as computer systems, communication systems, manufacturing systems, traffic systems etc. Discrete Event System (DES) framework has been widely used to model different aspects of these systems. The reason is the simplicity of the framework and the fact that most of the real world systems can be modeled as DES. Failure Detection and Diagnosis(FDD) theory of DES has been used to decide whether a system is working normally or not, if fault occurred which fault, what is the probability of occurrence of a fault etc.

6.1 Summary of contribution of the thesis

In this thesis, we have devised mechanisms for detecting attacks which do not have clear signature or anomaly pattern in Data Link layer, Network Layer and Transport layer. We have adopted FDD theory of DES to detect the different attacks across layers. The chapter wise contributions are as follows.

Contribution of chapter 2: ARP spoofing is a major attack in Data Link layer which is used to launch other attacks like, man-in-the-middle(MiTM), denial of service(DoS) etc. In this contribution we proposed a mechanism to detect this attack using active DES based IDS. The scheme uses an active probing mechanism based on ARP requests and responses, so it does not violate the principles of network layering architecture. Further, this being a software based approach which runs in one host (IDS), does not require any additional hardware or software patching in the hosts.

However, there are certain attacks like ICMP based attacks, where some of the traces in the model have uncontrollable events that are required to differentiate attack traces from normal traces. In this case, sending of probes and analyzing their responses are required for detecting attacks. Unlike ARP, in case of ICMP, sending probes and receiving their responses depend on congestion in the network. As congestion is an uncontrollable event, so sometimes traces under attack condition cannot be differentiated from the normal condition. This makes the attack non diagnosable in some cases (when congestion is present). In the next contribution, we explored detecting ICMP attacks by using the concept of partial diagnosis. In partial diagnosis only those paths which lead to attack certain states are considered and the uncertain paths are eliminated.

Contribution of chapter 3: Many of the attacks in the Network layer like connection reset, MiTM and DoS can be initiated with the exploitation of ICMP, an essential protocol in Network Layer. In this contribution an active detection mechanism to detect ICMP based attack is proposed. The genuinity of network traffic is verified by sending suitable probe packets to the hosts and validating their responses.

For ICMP attacks complete diagnosis of all paths are not possible due to inherent uncertainties (due to congestion) of network layer. In other words, there are paths in the DES model which lead to uncertain states about the presence of an attack. To deal with this, I-diagnosis framework has been adopted where indicator events are defined and

failure diagnosis is tested only in paths where a fault is followed by an indicator event. The detector in I-diagnosis framework may contain many redundant states that do not help in diagnosis purpose. So a reduced *I*-detector (called *RI*-detector) is also proposed which has less complexity. The detection scheme is successfully validated in a testbed with various attack scenarios and the results show the effectiveness of the proposed technique.

In case of attacks like Low Rate TCP DoS attack, normal and attack conditions do not differ clearly but they differ with some probability. Therefore I-diagnosis as well as active state based framework can not be used. In next contribution, we explored detection of such types of attacks by diagnosing sequence of events along with their probability of occurrence.

Contribution of chapter 4: Induced Low rate TCP-targeted DoS attack causes degradation of service and DoS by exploiting optimistic acknowledgement feature of TCP. As TCP constitutes bulk of the traffic in Transport layer, the effect of the attack is widespread. This Transport layer attack is relatively new and effective detection as well as mitigation mechanisms are not available so far. In this contribution a novel scheme has been proposed which reduces random bytes from a random TCP segment to verify the authenticity of optimistic acknowledgements.

To design a DES based IDS for Induced Low Rate TCP DoS attack, distinguishing normal and attack events is neither possible by active diagnosis nor by I-diagnosis framework because the attack and genuine traces may not clearly differ, but may differ with some probability. Stochastic DES framework has been adapted for detecting Induced low rate TCP attack where attack case can only be identified with some probability. The detection mechanism is verified in a test bed and results illustrated high detection rate and 100% accuracy with little compromise in TCP throughput.

In the above three contributions, state based DES modeling technique is used where the normal and attack models are constructed manually. But modeling complex attacks can be difficult in such a framework and may be prone to errors. In the last contribution, we explored Linear Time Temporal Logic(LTL) based DES to model a set of complex attacks based on NDP of IPv6.

Contribution of chapter 5: Attacker exploits NDP of IPv6 as it is stateless and lacks authentication by default. Although cryptographic mechanisms have been proposed to mitigate the attacks, they are not scalable and suffer from bootstrap problem. In this

contribution, we proposed an attack detection mechanism for NDP based attacks. As NDP based attacks are complex in nature, building DES based model manually is a challenging and error prone task. To enable verifiable and easy formal modeling of the IDS for NDP related attacks, LTL based DES framework is used. The IDS does not violate the standard NDP because the only additional requirement is probes, which are nothing but NDP related packets. Experimental results illustrated high attack detection rate and accuracy of 100% for NDP related attacks. Resource overheads were also minimal compared to modern network and system capacities.

6.2 Scope of Future Work

The work presented in this thesis justified the application of DES framework for detecting attacks at different layers, which are not detectable by standard signature or anomaly IDSs. The following are some of the possible future research directions.

- At present, different DES frameworks are required to detect attacks of different layers. Effort may be given to develop an unified DES framework that may deal with attacks of all the layers.
- As different IDSs are proposed for detecting attacks of different layers, their deployment are in different parts of the network so that they can get the corresponding layer's traffic. For effective attack detection there is need of collaboration or interaction among these IDSs. In future, it may be explored if the proposed layered IDSs can collaborate or work in a distributed manner to achieve a better detection capability.
- As wireless and mobile networks are getting popular, applicability of DES based IDS for wireless environment like Mobile Adhoc Network, Wireless Mesh Network etc. may be studied.

Bibliography

[1] CERT statistics. http://www.cert.org/stats. Accessed: 2013-11-20. 1

[2] M. G. Gouda and C. T. Huang. CSI/FBI computer crime and security survey. *Computer Security Journal*, XV(2), 1999. 1

[3] R. Heady, G. Lugar, M. Servilla, and A. Maccabe. The architecture of a network level intrusion detection system. Technical report, Department of Computer Science, University of New Mexico, 1990. 1

[4] M. Roesch. SNORT - lightweight intrusion detection for networks. In *USENIX System Administration Conference*, pages 229–238, 1999. 1.1.1, 2.2.2.4, 3.2.2.1

[5] P. A. Porras and P. G. Neumann. EMERALD: event monitoring enabling responses to anomalous live disturbances. In *National Information Systems Security Conference*, pages 1–13, 1997. 1.1.1

[6] V. Chandol, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Survey*, 41(3):1–58, 2009. 1.1.2

[7] A. Patcha and J. Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470, 2007. 1.1.2

[8] C. M. Kozierok. *TCP/IP Guide*. 1 edition, October 2005. 1.2.1, 2.1, 2.2.2.1, 4.1

[9] Cisco Systems. Cisco nexus 7000 series nx-os security configuration guide, release 4.1 configuring port security. http://www.cisco.com/en/US/docs/switches/datacenter/sw/4_1/nx-os/security/configuration/guide/sec_portsec.pdf. Accessed: 2013-11-20. 1.2.1, 2.1, 2.2.2.2

[10] Arpwatch. http://www.arpalert.org. Accessed: 2013-11-20. 1.2.1, 2.1, 2.2.2.3

[11] Arpdefender. http://www.arpdefender.com. Accessed: 2013-11-20. 1.2.1

[12] Colasoft-capsa. http://www.colasoft.com. Accessed: 2013-11-20. 1.2.1, 2.1, 2.2.2.3

[13] C. L. Abad and R. I. Bonilla. An analysis on the schemes for detecting and preventing ARP cache poisoning attacks. In *International Conference on Distributed Computing Systems Workshops*, pages 60–67, 2007. 1.2.1, 2.1, 2.2.2.4, 2.2.2.5

[14] V. Ramachandran and S. Nandi. Detecting ARP spoofing: An active technique. In *International Conference on Information Systems Security*, pages 239–250, 2005. 1.2.1, 2.2.2.6

[15] Z. Trabelsi and K. Shuaib. Man in the middle intrusion detection. In *Global Telecommunications Conference, 2006*, pages 1–6, 2006. 1.2.1, 2.2.2.6

[16] D. Thorsley and D. Teneketzis. Intrusion detection in controlled discrete event systems. In *Conference on Decision and Control*, pages 6047–6054, 2006. 1.2.1, 2.1, 2.2.2.6

[17] N. Hubbali, S. Biswas, S. Roopa, R. Ratti, and S. Nandi. LAN attack detection using discrete event systems. *ISA Transactions*, 50(1):119–130, 2010. 1.2.1, 2.1, 2.2.2.6, 2.5.2

[18] N. Hubbali, S. Biswas, S. Roopa, R. Ratti, S .Nandi, F.A. Barbhuiya, A. Sur, and V. Ramachandran. A DES approach to intrusion detection system for ARP spoofing attacks. In *Mediterranean Conference on Control and Automation*, pages 695–700, 2010. 1.2.1, 2.1, 2.2.2.6

[19] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1 edition, 1999. 1.2.1, 1.2.3, 2.2.2.6, 2.3.1, 4.1

[20] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D.C. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transaction on Automatic Control*, 40(9):1555 – 1575, 1995. 1.2.1, 1.2.2, 1, 1b, 1.2.4, 1.3, 2.2.2.6, 2.4.2.3, 2.4.2.4, 3.1, 3.2.2.4, 3.3.5, 3.4.3, 5.1

[21] M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event systems. *IEEE Transactions on Automatic Control*, 43:908–929, 1998. 1, 1.2.4, 2.2.2.6, 2.4, 2.4.2.3, 3.2.2.4, 5.1

[22] J. Postel. Internet control message protocol. RFC 792, Internet Engineering Task Force, September 1981. 1.2.2, 3.2.1.1, 3.2.1.2, 3.2.1.3, 3.2.1.5

[23] Cisco IOS Security Configuration Guide. http://www.cisco.com/en/US/docs/ios/security/configuration/guide/12_4/sec_12_4_book.html. Accessed: 2013-11-20. 1.2.2, 3.2.2.1

[24] H3C Technologies. Attack prevention technology white paper. http://www.h3c.com/portal/products___solutions/technology/ipv4___ipv6_services/technology_white_paper/200908/643656_57_0.htm. Accessed: 2013-11-20. 1.2.2, 3.2.2.1

[25] H. Chau. Network Security–Defense Against DoS/DDoS Attacks. http://www.infosecwriters.com/text_resources/pdf/Defense_DDoS.pdf, 2003. Accessed: 2013-11-20. 1.2.2, 3.2.2.2

[26] H. Lee, V. Thing, Y. Xu, M. Ma. ICMP Traceback with Cumulative Path, an Efficient Solution for IP Traceback . *LNCS, Information and Communications Security*, 2836:124–135, 2003. 1.2.2, 3.2.2.3

[27] J. Zhang, J. Liu, Z. Xu, J. Li, X. Ye . TRDP : a Trusted Router Discovery Protocol. In *International Symposium on Communications and Information Technologies ISCIT '07.*, pages 660 – 665, 2007. 1.2.2, 3.2.2.4

[28] A Kuzmanovic and E. W. Knightly. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 75–86, 2003. 1.2.3, 4.2, 4.2

[29] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *International Conference on Network Protocols*, pages 192–201, 2001. 1.2.3, 4.2

[30] A. Kuzmanovic and E. W. Knightly. Low-rate TCP-targeted denial of service attacks and counter strategies. *IEEE/ACM Transactions on Networking*, 14(4):683–696, 2006. 1.2.3, 4.2

[31] H. Sun, J. C. S. Lui, and D. K. Y. Yau. Defending against low-rate TCP attacks: Dynamic detection and protection. In *International Conference on Network Protocol*, pages 196–205, 2004. 1.2.3, 4.2

[32] A. Shevtekar, K. Anantharam, and N. Ansari. Low rate TCP denial-of-service attack detection at edge routers. *IEEE Communications Letters*, 9(4):363–365, 2005. 1.2.3, 4.2

[33] Y. Kwok, R. Tripathi, Y. Chen, and K. Hwang. Hawk: halting anomalies with weighted choking to rescue well-behaved TCP sessions from shrew ddos attacks. In *International Conference on Networking and Mobile Computing*, pages 423–432, 2005. 1.2.3, 4.2

[34] Xiapu Luo and Rocky K. C. Chang. On a new class of pulsing denial-of-service attacks and the defense. In *Network and Distributed System Security Symposium*, pages 61–79, 2005. 1.2.3, 4.2

[35] C. Chang, S. Lee, B. Lin, and J. Wang. The taming of the shrew: mitigating low-rate TCP-targeted attack. *IEEE Transactions on Network and Service Management*, 7(1):1–13, 2010. 1.2.3

[36] D. Thorsley and D. Teneketzis. Diagnosability of Stochastic Discrete-Event Systems. *IEEE Transaction on Automatic Control*, 50(4):476–492, April 2005. 1.2.3, 3.6, 4.1, 4.2, 4.3, 4.6

[37] S. Thomson, T. Narten, and T. Jinmei. IPv6 stateless address autoconfiguration. RFC 4862, Internet Engineering Task Force, September 2007. 1.2.4, 5.2.1.2

[38] S. A. Kent and R. Atkinson. Security architecture for the Internet Protocol. RFC 2401, Internet Engineering Task Force, November 1998. 1.2.4

[39] D. Harkins and D. Carrel. The internet key exchange (IKE). RFC 2409, Internet Engineering Task Force, November 1998. 1.2.4

[40] Ed. J. Arkko, J. Kempf, B. Zill, and P. Nikander. SEcure Neighbor Discovery (SEND). RFC 3971, Internet Engineering Task Force, March 2005. 1.2.4, 3.2.2.4, 5.2.2

[41] Ed. J. Arkko, J. Kempf, B. Zill, and P. Nikander. Cryptographically Generated Addresses (CGA). RFC 3972, Internet Engineering Task Force, March 2005. 1.2.4, 5.2.2

[42] Ndpmon. http://ndpmon.sourceforge.net. Accessed: 2013-11-20. 1.2.4, 5.2.2

[43] S. Jiang and R. Kumar. Failure Diagnosis of Discrete-Event Systems with Linear-Time Temporal Logic specifications. *IEEE Transactions on Automatic Control*, 49(6):934 – 945;, 2004. 1.2.4, 5.1, 5.3, 5.3.2.2, 5.4, 5.6

[44] M. Izadi, M. Bonsangue, and D. Clarke. Buchi automata for modeling component connectors. *Software & Systems Modeling*, 10(2):183–200, 2011. 1.2.4, 5.1

[45] Sean Whalen. An Introduction to ARP Spoofing. Technical report, April 2001. Accessed: 2013-11-20. 2.1

[46] Han-Wei Hsiao, Cathy S. Lin, and Ssu-Yang Chang. Constructing an arp attack detection system with SNMP traffic data mining. In *International Conference on Electronic Commerce*, pages 341–345, 2009. 2.1, 2.2.2.4

[47] Mohamed G. Gouda and Chin-Tser Huang. A secure Address Resolution Protocol. *Computer Networks*, 41(1):57–71, 2003. 2.1, 2.2.2.5

[48] Wesam Lootah, William Enck, and Patrick McDaniel. TARP: Ticket-based address resolution protocol. In *Annual Computer Security Applications Conference*, pages 106–116, 2005. 2.1, 2.2.2.5

[49] J. C. Mogul, Christopher A. Kent, C. Partridge, and K. McCloghrie. IP MTU discovery options. RFC 1063, Internet Engineering Task Force, July 1988. 3.2.1.1

[50] F. Gont. ICMP attacks against TCP. RFC 5927, Internet Engineering Task Force, 2010. 3.2.1.1, 3.2.1.1, 3.2.1.2

[51] M. Baltatu, A. Lioy, F. Maino, and D. Mazzocchi. Security issues in control, management and routing protocols. *Computer Networks*, 34(6):881 – 894, 2000. 3.2.1.4

[52] Ping. http://www.ping127001.com/pingpage.htm. Accessed: 2013-10-17. 3.2.1.5

[53] http://www.sans.org/rr/threats/ICMPattacks.php. Accessed: 2011-05-17. 3.2.1.5

[54] Cisco ios netflow. http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html. Accessed: 2013-10-17. 3.3.1

[55] H. Chau. Simple Network Management Protocol. http://www.cisco.com/en/US/docs/internetworking/technology/handbook/SNMP.html. Accessed: 2011-07-12. 4

[56] How to calculate bandwidth utilization using snmp. http://www.cisco.com/en/US/tech/tk648/tk362/technologies_tech_note09186a008009496e.shtml. Accessed: 2013-10-17. 3.3.2.1

[57] Cisco Systems. Catalyst 3560 switch getting started guide. http://www.cisco.com/en/US/docs/switches/lan/catalyst3560/hardware/quick/guide/3560gsg_08.html. Accessed: 2013-11-20. 3.5, 5.5

[58] Quagga software routing suite. http://www.cert.org/stats/cert_stats.html. Accessed: 2013-10-17. 3.5

[59] Net-snmp. http://www.cert.org/stats/cert_stats.html. Accessed: 2012-09-15. 3.5

[60] S. M. Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communications Review*, 19(2):32–48, 1989. 4.1

[61] V. Jacobson. Congestion avoidance and control. *SIGCOMM Computer Communication Review*, 18(4):314–329, August 1988. 4.1

[62] V. Kumar, P. Jayalekshmy, G. Patra, and R. Thangavelu. On remote exploitation of TCP sender for low-rate flooding denial-of-service attack. *IEEE Communications Letters*, 13(1):46–48, 2009. 4.2, 4.2

[63] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP congestion control with a misbehaving receiver. *SIGCOMM Computer Communication Review*, 29:71–78, 1999. 4.2, 4.2

[64] R. Sherwood, B. Bhattacharjee, and R. Braud. Misbehaving TCP receivers can cause internet-wide congestion collapse. In *Conference on Computer and Communications Security*, pages 383–392, 2005. 4.2

[65] Microsoft Technet. IPv6 features. http://technet.microsoft.com/en-us/library/cc738636(v=ws.10).aspx. Accessed: 2012-07-05. 5.1

[66] A. Cimatti, E.M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An open source tool for symbolic model checking. In *International Conference on Computer Aided Verification*, pages 359–64, 2002. 1.2.4, 5.1, 5.4.3.3

[67] T. Narten, E. Nordmark, and W. Simpson. Security features in IPv6. Whitepaper, SANS Institute, 2002. 5.2.2

[68] H. Rafiee, A Alsa'deh, and C Meinel. Winsend: Windows secure neighbor discovery. International Conference on Security of Information and Networks, pages 243–246, 2011. 5.2.2

[69] M. Huth and M.Ryan. *Logic in Computer Science*. Cambridge University Press, 1 edition, 2004. 5.3.1

[70] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1 edition, 1999. 5.3.1

[71] P. Wolper. The tableau method for temporal logic: An overview. In *Logique et Analyse*, pages 122–127, 1985. 5.3.3

Publications out of the work

International journals

1. F. A. Barbhuiya, G. Bansal, N. Kumar, S. Biswas and S. Nandi " Detection of Neighbor Discovery Protocol based attacks in IPv6 Network", International Journal of Networking Science, Vol 2, Issue 3, pages 91-113, Springer

2. P. Banerjee, M. Mitra, F. A. Barbhuiya, S. Biswas and S. Nandi "IDS for ARP Spoofing using LTL based Discrete Event System Framework", International Journal of Networking Science, Vol 2, Issue 3, pages 114-134, Springer

3. F.A.Barbhuiya, S. Biswas, S. Nandi, " An Active Host-Based Intrusion Detection System for ARP-Related Attacks and its Verification", International Journal of Network Security & Its Applications. Vol 3, No 3, pp 163-180

International conferences

1. F.A. Barbhuiya, S. Roopa, R. Ratti, S. Biswas, S. Nandi"An Active Detection Mechanism for Detecting ICMP Based Attacks", 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2012), Liverpool,UK, pages 51-58 (IEEE)

2. F.A. Barbhuiya, V. Gupta, S. Biswas, S. Nandi " Detection and Mitigation of Induced Low Rate TCP-Targeted Denial of Service Attack", 6th International Conference on Software Security and Reliability (SERE 2012), Washington DC, USA, pages 291-300 (IEEE)

3. F.A. Barbhuiya, S. Biswas, S. Nandi "An active detection mechanism for NDP attacks in IPv6", International Conference on Security of Information and Networks (SIN 2011), Sydney, Australia, pages 111-118 (ACM SIG-SAC).

4. F.A. Barbhuiya, S. Biswas, S. Nandi "An active DES based IDS for Detecting ARP Spoofing", International Conference on Systems, Man, and Cybernetics (SMC 2011), Alska, USA, pages 2743-2748 (IEEE).

Other publications of the author

International journals

1. S. Chakraborty, A. Rai, F. A. Barbhuiya, A. Sur, S. Biswas and S. Nandi "Topology Adaptive Computation of Distributed IDS Set for Detecting Attacks on STP", Journal of Information Assurance and Security vol 7, issue 4, pages 284-295

International conferences

1. G. Bansal, N. Kumar, F. A. Barbhuiya, S. Biswas, S. Nandi, "Scalable Implementation of Active Detection Mechanism for LAN Based Attacks", International Conference on Network Security & Applications (CNSA 2011), Chennai,pp 258 - 267 (CCIS)

2. N. Hubballi, S. Biswas, F.A.Barbhuiya, Roopa S., R. Ratti, S. Nandi, " Discrete Event System Approach to Intrusion Detection System for ARP Attacks" Mediterranean Conference on Control and Automation-2010 (MED 2010), Marrakech, Morocco, pp 695 - 700 (IEEE).

3. N. Hubballi, Roopa S., R. Ratti, F.A. Barbhuiya, S. Biswas, S. Nandi, A. Sur, V. Ramachandran, "An Active Intrusion Detection for LAN Specific Attacks", International Conference on Information Security and Assurance-2010 (ISA 2010), Miyazaki, Japan, pp 129-142 (LNCS).

4. F.A. Barbhuiya, N Hubbali, S. Biswas, S. Nandi "Completeness of LAN Attack Detection using Discrete Event Systems", International Conference on Network Security & Applications (CNSA 2011), Chennai, India, pages 131-139 (CCIS)

5. F.A. Barbhuiya, S. Biswas, N. Hubballi, S. Nandi, "A Host Based DES Approach for Detecting ARP Spoofing", IEEE Symposium on Computational Intelligence in Cyber Security 2011 (CICS 2011), Paris, France, pages 114-121 (IEEE).

6. F.A. Barbhuiya, Roopa S., R. Ratti, N. Hubballi, S. Biswas, S. Nandi, A. Sur and V. Ramachandran, "An Active Host-based Detection Mechanism for ARP-related Attacks", International Conference on Network and Communications Security-2010 (NCS 2010), Chennai, India, pp- 432-441 (CCIS).

CURRICULUM VITAE

Name : Ferdous Ahmed Barbhuiya

Father's Name : Sajjadur Rahman Barbhuiya,

Address : Department of Computer Science and Engineering,
IIT Guwahati,
Assam 781039, India

Email: ferdous@iitg.ac.in

Ferdous Ahmed Barbhuiya obtained his Bachelors of Engineering (BE) degree in Computer Science and Engineering from Jorhat Engineering College under Dibrugarh University, Jorhat, Assam in the year 2001 and Master of Technology in Computer Science and Engineering in the year 2007 from Indian Institute of Technology Guwahati. He was placed in the 8^{th} position in the university in his BE. He has worked as a Scientific Officer in IIT Guwahati from June 2002 to December 2007. He has also worked in GlobalLogic Inc. from December 2007 to June 2009 as a Lead Engineer and there after in IIT Guwahati as Project Fellow. His research interests include system and network security, Network Protocol and Sensor Fusion etc. He has authored 21 research papers till date.