Formal Specification and Verification of WiFiRe protocol

Dissertation

submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

by

Ch Sudheer Keshav

(Roll no. 05329008)

under the guidance of

Prof. Sridhar Iyer and

Prof. Krishna S

Kanwal Rekhi School of Information Technology

Indian Institute of Technology Bombay

2007

Dissertation Approval Sheet

This is to certify that the dissertation entitled

Formal Specification and Verification of WiFiRe

by

Ch Sudheer Keshav

(Roll no. 05329008)

is approved for the degree of **Master of Technology**.

Prof. Sridhar Iyer

(Supervisor)

Prof. Krishna S

(Co-Supervisor)

Prof. Anirudha Sahoo

(Internal Examiner)

Dr. Vijay T Raisinghani

(External Examiner)

Prof. V M Gadre

(Chairperson)

Date: _____

Place: _____

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
CERTIFICATE OF COURSE WORK

This is to certify that **Mr. Ch Sudheer Keshav** was admitted to the candidacy of the M.Tech. Degree and has successfully completed all the courses required for the M.Tech. Programme. The details of the course work done are given below.

Sr.No.	Course No.	Course Name	Credits
		Semester 1 (Jul – Nov 2005)	
1.	HS699	Communication and Presentation Skills (P/NP)	4
2.	IT601	Mobile Computing	6
3.	IT605	Computer Networks	6
4.	IT619	IT Foundation Lab	8
5.	IT623	Foundation course of IT - Part II	6
6.	IT608	Datamining and Data Warehousing	6
		Semester 2 (Jan – Apr 2006)	
7.	CS640	Formal Language and Models for Natural Computing	6
8.	HS618	Introduction to Indian Astronomy	6
9.	IT610	QoS in Networks	6
10.	IT680	Systems Lab.	6
11.	IT694	Seminar	4
		Semester 3 (Jul – Nov 2006)	
12.	CS601	Algorithms and Complexity	6
13.	CS615	Formal Specification and Verification of Programs	6
		M.Tech. Project	
14.	IT696	M.Tech. Project Stage - I (Jul 2006)	18
15.	IT697	M.Tech. Project Stage - II (Jan 2007)	30
16.	IT698	M.Tech. Project Stage - III (Jul 2007)	42

I.I.T. Bombay Dy. Registrar(Academic)

Dated:

Acknowledgements

I would like to thank my guide **Prof. Sridhar Iyer**, for his invaluable support, encouragement and guidance. I also take this opportunity to thank **Prof. Krishna S**, **Prof. Anirudha Sahoo**, for their precious suggestions throughout this project work.

I would like to thank my lab-mates for helpful discussions through out my project, and the KReSIT department for providing me world class computing infrastructure.

I would also like to thank my **family** and **friends**, who have been a source of encouragement and inspiration throughout the duration of the project.

I would also like to thank the entire KReSIT family for making my stay at IIT Bombay a memorable one.

Ch. Sudheer Keshav

I. I. T. Bombay

July 10^{th}, 2007

Abstract

WiFiRe stand for **WiFi-Rural Extension**[1]. There has been a lot of growth in Internet as well as cellular telephony, but it is mostly restricted to only cities in developing countries like India. A number of alternative solutions have been proposed which in theory are suitable for rural areas, but because of their high cost WiFiRe is the best option. WiFiRe is a mixture of the advantages of both 802.11[2] and 802.16[3]. The cost affordability of 802.11 and QoS capabilities of 802.16 are merged to obtain WiFiRe. The WiFiRe MAC is time-division duplex (TDD) over a single 802.11b channel along with a multi-sector TDM mechanism. WiFiRe will be able to provide long range communication by dividing the whole area into sectors, each sector will be having one base station which is directional so that it covers more distance.

Formal methods may be used to give a description of the system to be developed, at whatever level(s) of detail desired. This formal description can be used to guide further development activities; additionally, it can be used to verify that the requirements for the system being developed have been completely and accurately specified. Formal methods are mathematically-based techniques for the specification, development and verification of software and hardware systems. Formal methods and testing are a perfect couple, testing and formal verification are both necessary. A formally verified specification is a good starting point for testing. So a formal specification and verification of WiFiRe is required.

Contents

List of Figures

Abbreviations

Abbreviations

CID : Connection Identifier

PoP : fiber Point of Presence

BS : Base Station

ST : Subscriber Station

DSA : Dynamic Service Addition

DSD : Dynamic Service Deletion

DSC : Dynamic Service Change

TDD : Time Division Duplex

TDM : Time Division Multiple access

TDD-MSTDM : Time Division Duplex, multi sector TDM

DL-MAP : Downlink MAP

UL-MAP : Uplink MAP

UGS : Unsolicited Grant Service

rtPS : Real Time Polling Service

nrtPS : non-real Time Polling Service

SAP : Service Access Point

REQ : Request

TO : Timeout

PHY : Physical Layer

PDU : Protocol Data Unit

SDU : Service Data Unit

LTL : Linear Temporal Logic

Chapter 1

Introduction and Motivation

Over the last decade there has been almost an exponential growth of the Internet as well as cellular telephony. Much of the development was present in developed countries and in developing countries like India their growth has been limited to metro cities. About 70 percent of India's population live in its villages, most of which are in the plains. The average village has 250-300 households, and occupies an area of 5 sq. km.[1] Most of this is farmland, and typically the houses are in one or two clusters. Villages are thus spaced 2-3 km apart and the average village occupies an area of 5 sq. km., and spread out in all directions from the market centers. The market centers are typically spaced 30-40 km apart. Each such center serves around 250-300 villages, in a radius of about 20 km . The telecommunication backbone network and the base stations of the mobile (cellular) operations are networked using optical fiber. However, all these end abruptly at towns. Even though fixed wireless telephones are provided to many villages, but broadband internet access has not reached the rural areas even now. **WiFiRe** because of the license free nature of the WiFi spectrum (IEEE 802.11b, 2.4 GHz Band) and the easy availability of WiFi RF chipsets is suitable to provide long-range communications (15-20 Kms) for rural areas. WiFiRe is meant for a star topology - a Base Station (BS) at the fiber Point of Presence and Subscriber Terminals (ST) in the surrounding villages with sectorized antennas at the BS and a directional antenna at each ST.

In India, where the telecom infrastructure is poor and last-mile connections are typically through copper cable, DSL and fiber optic, installation costs are high as it requires ripping up streets to lay cables. The ability to provide these connections through wireless, without laying wire or cable in the ground, greatly lowers the cost of providing these services. This is why WiFiRe is an attractive alternative for providing broadband services in rural India.

In developing countries that lack a well-developed wired infrastructure, WiFiRe offers a way to extend broadband Internet service to many different parts of the country. WiFiRe could thus bring broadband access into the homes and businesses of millions of people in rural areas.

1.1 Motivation

The need for formal specification systems has been noted for years, *e.g* Backus normal form is used as formal notation for describing programming language syntax. The use of formal methods for software and hardware design is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analysis can contribute to the reliability and robustness of a design. Formal Specification helps build more robust, more maintainable software with fewer bugs and defects. A distinguishing feature of high assurance systems is that they are modeled mathematically using formal methods. This modeling enables formal reasoning about the system design, that can be used to prove that the system has certain properties. We are using Model checking, an automated theorem proving technique in which a system verifies certain properties by means of an exhaustive search of all possible states that a system could enter during its execution. Through a formal verification we can find the potential problems associated with the protocol before actually deploying it.

1.2 Problem Description

A formal specification of WiFiRe is required because it captures the essential properties of the problem, provides precise guidelines for someone who wants to implement this protocol, and provide a measure against which other similar protocols can be checked for correctness. The specification can be viewed as a "Black Box" which has a user interface that gets all the inputs that the protocol receive and sends out all the outputs that we want the protocol to produce.

The aim of this project is to formally specify and verify the protocol. The aim also includes choosing a proper tool to serve the purpose. We used Spin[4], a model checking tool for this purpose. Through formal verification, we are tried to find any potential

problems associated with the protocol, to find if certain properties are satisfied, to find if the protocol is free from deadlocks and livelocks before actually deploying it. Our part includes specifying and verifying properties related to following phases of the protocol.

- **Network Initialization sub-procedures:** This part includes Ranging, Registration.
- **Connection management sub-procedures:** This part includes, service addition, service change, and service deletion.
- **Data transport sub-procedures:** This part includes fragmentation and re-assembly of data.
- **Periodic Ranging**.
- **QoS related sub-procedures**.

1.3 Outline of the report

The remaining part of the report is arranged as follows: chapter 2 describes briefly, the various alternative technologies to WiFiRe and the reasons as to why we have chosen Spin as our verification tool, chapter 3 contains the overview of WiFiRe with detailed state transition diagrams, chapter 4 describes the modeling of WiFiRe in PROMELA, the modeling language for Spin, chapter 5 describes the various properties verified and the remedies provided to the problems persistent in the protocol, and finally chapter 6 concludes the report.

Chapter 2

Literature Survey

2.1 Technology Alternatives to WiFiRe

As part of literature survey, we have studied various alternative technologies to WiFiRe. These include WIMAX(802.16)[3], WiFi(802.11)[2], Digital Gangetic Plain Project(DGP)[5], and cellular technologies(GSM, GPRS, and CDMA)[1].

A summary of the study is as follows:

1. Present day mobile cellular technologies (such as GSM, GPRS, CDMA) may meet the cost targets but are unlikely to be able to provide broadband services as defined above.

2. Proprietary broadband technologies (such as iBurst, Flash-OFDM) meet many of the performance requirements, but typically have low volumes and high costs. The indigenously developed Broadband corDECT technology of Midas Communication Technologies, Chennai is a fixed-access wireless broadband system that meets the performance and cost requirements.

3. WiMAX-d (IEEE 802.16d), is a new standards-based technology for fixed wireless-access that meets the performance requirements, but not the cost targets.

4. WiFi (IEEE 802.11b), is an inexpensive local-area broadband technology. It can provide 256 kbps or more to tens of subscribers simultaneously, but can normally do so only over short distances (less than 50 m indoors). One attraction of WiFi technology is the de-licensing of its spectrum in many countries, including India. Another is the availability of low-cost WiFi chipsets.

5. A similar approach is Digital Gangetic Plains (DGP) project[5]. They have built a testbed in a rural setting consisting of multi-hop directional 802.11 links, the testbed spanning up to 80km at its longest. Main disadvantages of this project include this is not ad-hoc, more computation power needed at each access point as each act as a router, and it doesn't give QoS guarantee

WiFiRe, as defined herewith, is one such alternative MAC designed to leverage the low cost of WiFi technology for providing fixed wireless access. It is a Time Division Duplex (TDD) communication protocol over a single WiFi channel, along with a multi-sector Time Division Multiplex (TDM) mechanism. This is explained in the chapter 3.

2.2 Linear Time Model Checking

Formal logic can be used to describe precisely many practical forms of specification, including, safety, liveness, and properties. The most widely studied are temporal logics which were introduced for specification and verification purposes in computer science. Temporal logics support the formulation of properties of system behavior over time. Different interpretations of temporal logics exist depending on how one considers the system to change with time. Linear temporal logic allows the statement of properties of execution sequences of a system. Branching temporal logic allow the user to write formulas which include some sensitivity to the choices available to a system during its executions. It allows the statement of properties of about possible execution sequences that start in a state.

In this report we consider only specification expressed using the LTL[6] linear time logic. This logic is formally based on finite state models where at each moment there may be several different possible futures. Technically speaking linear time boils down to the fact that from the initial state the system can have a set of different possible execution sequences. The underlying notion of the semantics of a linear time temporal logic is thus a set of infinite sequences. Each sequence is intended to represent a single possible computation. The set of sequences itself thus represents all possible computations.

The syntax of linear time logic (**LTL**) is defined as follows. The most elementary expressions in LTL are atomic propositions. The set of atomic propositions is denoted by AP with typical elements p, q and r. We define the syntax of LTL in Backus-Naur Form. For $p \in AP$ the set of LTL formulas is defined by

$$\phi ::= \top | p | \neg\phi | \phi \wedge \varphi | X\phi | \phi U \varphi$$

where X and U are linear temporal operators that express a property over a single path. Other frequently used derived operators include G (always) and F future. In literature the symbols $\Box$ ([]) and $\Diamond$ (<>) are sometimes used to denote G and F, respectively.

SPIN allows us to verify different properties of our models via assertion checking, cycle detection and satisfaction of LTL formulas.

2.3 Formal Verification

One of the problems in formal verification has long been that there is no generally agreed upon benchmark set that researchers can use to compare the performance of different algorithms, different implementation, or different verification tools.

There are several problems with the development of a standard approach. Different verification tools (such as Spin, SMV, $Mur\varphi$, or Uppaal) are typically designed for very specific domains of problems that do not always overlap. Spin, for instance, is designed for models of asynchronous software systems, SMV was designed primarily for the verification of hardware circuits, and Uppaal was designed primarily for real-time verification problems. Modeling a hardware circuit as a Spin model is typically not a good idea, as is the verification of a real-time asynchronous software verification problem in SMV.

Each verification tool tends to have its own specification format, and a specific set of primitives for which it can give optimized performance. Translation from one format into another is always plagued with the problem that an encoding that is efficient for one tool is translated into an encoding that is inefficient for another tool.

2.3.1 Verification Tools Used

As part of decision making on which tool to use, we have used two of the most used and easily available verification tools. These are $Mur\varphi$[7] and Spin. We have used $Mur\varphi$ for analysis of any deadlocks in TCP in connection establishment and connection termination phases. $Mur\varphi$ has a reduction technique called Symmetry Reduction which exploits the symmetry present in the system that is being modeled. A similar kind of mechanism called Partial Order Reduction is also present in Spin. If there is not enough space to store the states, the analysis of the system under verification fails. But so far, Spin is the only verification system that can avoid this trap. All other existing automated verification system (irrespective on which formalism they are based) simply run out of memory and abort their analysis without returning a useful answer to the user[4]. Spin realizes a partial coverage only in cases where traditional verifiers are either unable to perform a search, or realize a far smaller coverage. It has been proven that in no case will Spin produce an answer that is less reliable than that produced by other automated verification systems[4].

One of the advantage of $Mur\varphi$ is that it provides the option to write procedures. Such an option is not present in Spin. The reason for this restriction to the basic language is that Spin targets the verification of process interaction and process coordination structures, and not internal process computations. Abstraction is then best done at the process and system level, not at a computational level. It is possible to approximate a procedure call mechanism with Promela process instantiations. We have searched for parameterized verification in $Mur\varphi$. But this option is not provided in the recent versions of $Mur\varphi$. As the system we are going to model is complex for even the simplest of the scenarios and its complexity increases when we increase the number of STs and BSs, we have opted for Spin which has many reduction techniques which we can explore and verify our system for correctness with high probability.

2.3.2 Verification Tool A Simple Case-Study

There are many freely available formal method tools that can assist with the formal methods of software design. These are the results of the experiments conducted by Yifei Dong *et al.*[8] on i-protocol, a protocol that makes for a formidable case study for verification tools, using five of the widely used model checkers, namely, COSPAN, Murφ, SMV,

Spin, and XMC. SPIN and COSPAN ran out of memory, SMV's memory performance was good, but time taken was more. Murφ and XMC performed the best on i-protocol, completing on all cases of interest. XMC was however faster than Murφ.

Later it was proved[9] that with either default Spin verification running, or with a reasonable choice of parameter settings, the version of Spin used for tests in [8] can outperform the results obtained with XMC to some extent. If new version of Spin was used it would outperform XMC comprehensively. So of the above specified widely used verification tools, SPIN suits our purpose more than the rest.

The summary of the tool comparisons provided in SPIN website[4] is as shown below. Ths study is done based on a large benchmark set of verification models provided in BEEM[10] database. The collection contains 57 separately models, categorized in 8 different groups (communication protocols, mutual exclusion algorithms, leader election algorithms, other protocols, controllers, planning and scheduling, puzzles, and miscellaneous). Each model is parameterized to yield between 3 and 8 different problem instances, which brings the total number of models in the database to 298. 232 of the 298 models which had SPIN version were used in the study.

- Number of cases where Spin failed: 42 of 232 fraction: 0.181034.
- Number of cases where Beem[1] failed: 38 of 232 fraction: 0.163793
- Number of cases where Spin has shorter runtime: 166 of 232 fraction: 0.715517
- Number of cases where Beem has shorter runtime : 33 of 232 fraction: 0.142241
- Number of cases where Spin has fewer states: 79 of 232 fraction: 0.340517
- Number of cases where Beem has fewer states: 107 of 232 fraction: 0.461207

2.4 Spin

Spin is a powerful verification system based on model checking. Model checking is a method to algorithmically verify formal systems. This is achieved by verifying if the model,

[1]BEnchmarks for Explicit Model checkers

often derived from a hardware or software design, satisfies a formal specification. The specification is often written as temporal logic formulas. A model-checking tool accepts system requirements or design (called models) and a property (called specification) that the final system is expected to satisfy.

2.4.1 Success Stories Of Spin

One of the reasons for choosing Spin as our verification tool is its common application in modelling and verifying communication protocols. In the area of mobile communication, part of the MCNet(Mobile Communication Network)[11] architecture has been simulated and verified in Spin.. Deadlocks have been detected and many problems related to message exchange have been sorted out. Spin has also been for Formal verification of Ad-Hoc routing protocols[12].. The protocol is proved to be error free. Broadcast systems, Mobility and timers have been modelled. SPIN has been used for Modeling and Formal Verification of DHCP[13]. Some conditions like checking for unique IP, etc are similar to WiFiRe. Many other Network related protocols have also been formally verified using Spin[4].

2.4.2 Spin's Reduction Algorithms

The validators produced by SPIN are among the most fastest programs for exhaustive searching known.The Validators are used in two modes. For small and medium size models the validators can be used with an exhaustive state space. For systems that are larger, the validators can be used in supertrace mode, with bit state storage technique that can collapse the state space to a small number of bits per reachable state with minimal side-effect.

The following are the brief descriptions of some of the reduction algorithms[6] used by Spin.

- **Partial-Order Reduction**: (Helps in DFS algorithm used by SPIN) is one of SPINs primary weapons against the state explosion problem.
- **State vector compression**: State vector compression is present which helps in compression of the state vector.

- **Dead variable analysis**: Variable which can neither influence the control flow nor any variables which occur in atomic propositions.
- **Statement merging**: If there are two consecutive local invisible actions in the model then we can statically merge them into one atomic action.
- **Hash Compaction**: This technique tries to increase the size of hash table beyond what could be available on an average machine.
- **Bitstate Hashing**: With bit state storage technique we can collapse the state space to a small number of bits per reachable state with minimal side-effect.

2.4.3 Strengths of Spin

- With Java PathFinder, it is possible to translate programs written in JAVA version 1.0 to PROMELA.
- The tool is specifically designed to scale well, and to handle even very large problem sizes efficiently.
- SPIN offers the possibility to perform both simulations and verifications.
- Promela is enriched with a set of primitives allowing the creation and synchronization of processes, including the possibility to use both synchronous and asynchronous communication channels.
- A specification consists of one or more processes, each describing the behavior of one component of the system.
- Processes may communicate via channels by sending and receiving messages.
- Correctness claims to be checked are expressed as temporal logic formulas.
- Spin works on-the-fly, which means that it avoids the need to preconstruct a global state graph, or Kripke structure, as a prerequisite for the verification of system properties. i.e the automation need not be completely available before checking for emptiness.
- Spin can be used as a simulator, exhaustive verifier and proof approximation system.

Chapter 3

WiFiRe Protocol

A **WiFiRe**[**1**] system is one approach to design a long-range and low-cost wireless communication network. The WiFiRe physical layer (PHY) directly employs the low-cost WiFi PHY (IEEE 802.11b, Direct Sequence Spread Spectrum). The WiFi PHY is for operation in the 2.4 GHz band and designed for a wireless local area network (LAN) with 1 Mbps, 2 Mbps and 11 Mbps data payload communication capability. WiFiRe extends the transmission range of the WiFi PHY to 15-20 Kilometers, by using a deployment strategy based on sectorized and directional antennas. The WiFiRe MAC is time division duplexed, multi-sector TDM (TDD-MSTDM). The WiFiRe MAC is similar to the WiMax MAC (IEEE 802.16), in some respects. WiFiRe adopts a star topology. As shown in Figure 3.1, a WiFiRe system consist of a set of sectorized antennas at the base station (BS), mounted on a transmission tower with a height of 40 meters and directional antennas at the subscribers terminals (ST), mounted on poles with height of around 10 meters. Typically a system is designed to cover an approximately circular area with radius of 15 Kilometers, around the tower. This area is called as a Cell. The BS is connected to the external world (Internet) through the fiber PoP, while the ST is connected to voice and data terminals, through a local area network.

The main design goal of WiFiRe is to enable the development of low-cost hardware and network operations for outdoor communications in a rural scenario. This has two implications: (i) a WiFiRe avoids frequency licencing costs by operating in unlicensed 2.4 GHz frequency band, and (ii) WiFiRe uses WiFi physical layer(PHY), due to the low cost and easy availability of WiFi chipsets.

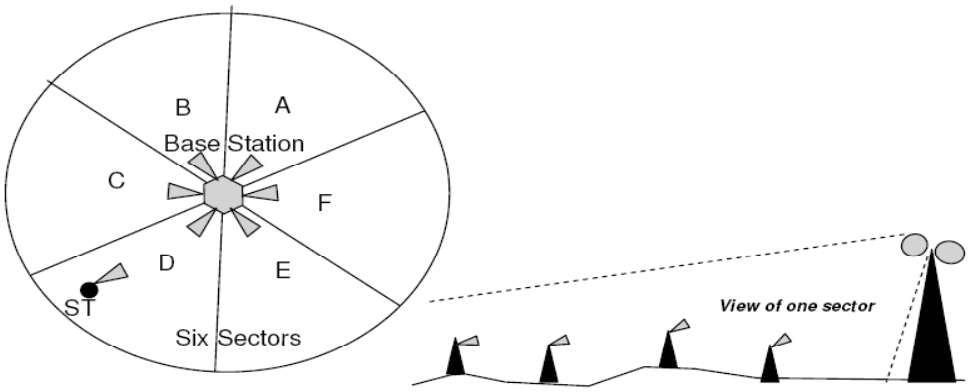

Figure 3.1: WiFiRe Topology[1]

3.1 WiFiRe Reference Model

WiFiRe reference model is as shown in the figure 3.2.

The SSS utilizes the services of provided by LCS and in turn provides services to external higher layers. The main functions of SSS include classification and processing(if required) of higher layer PDUs, mapping of higher layer PDUs into MAC SDUs.
The main services provided by LCS sublayer are connection provisioning, data transport. The security sublayer present as part of LCS takes care of authentication of end-points and secure transmission of the connection's PDUs.

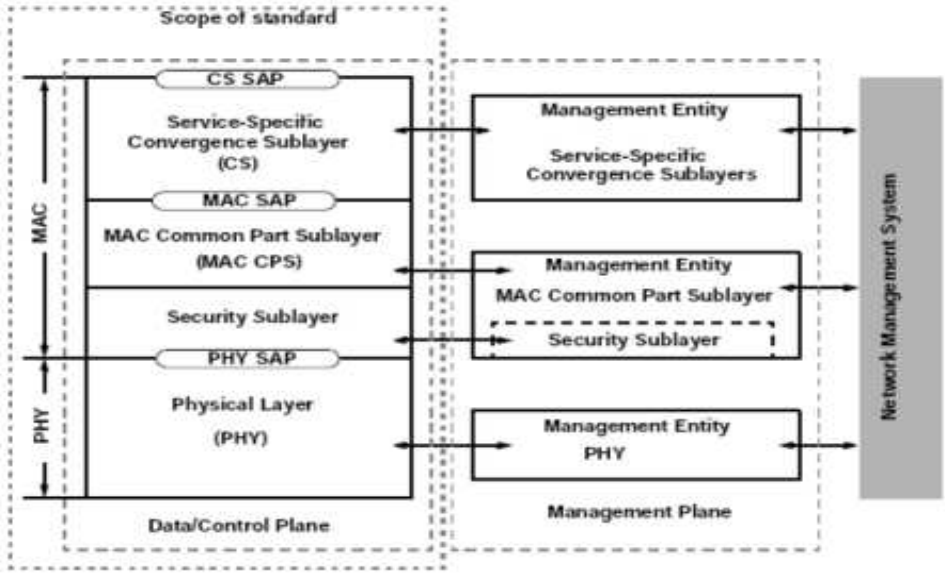

Figure 3.2: WiFiRe Reference Model[1]

3.2 WiFiRe MAC

The MAC mechanism is a time division duplexed, multi-sector TDM (TDD-MSTDM) scheduling of slots. Time is divided into frames. Each frame is further partitioned into a downlink (DL) and an uplink (UL) segment, which need not be of equal durations as shown in figure 3.3. The downlink from the system S to the ST(s) operates on a point-to-multipoint basis. The uplink from a ST to system S operates on a point-to-point basis. Within each segment there are multiple slots, of equal duration each.

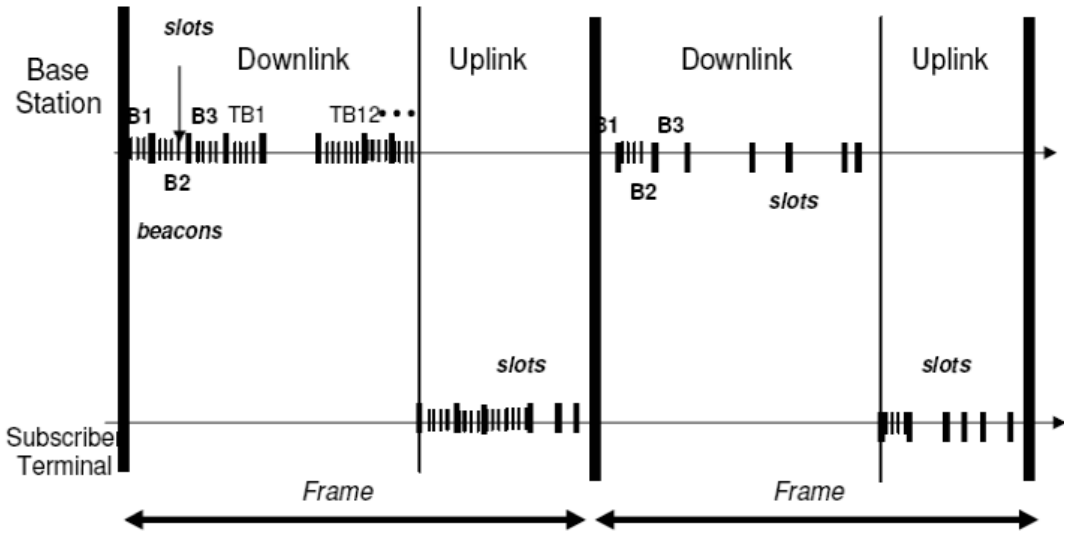

Figure 3.3: WiFiRe Multiple Access Mechanism[1]

3.2.1 Network Initialization

The association between a ST and a System S is static. But deciding on which BS to use for communication is done through ranging and registration. An ST should communicate with only one S.

3.2.1.1 Ranging

Upon completion of power-up sequence and self-initialization, Ranging is done by the ST in order to synchronize clock and other physical layer parameters with the system S. The System S periodically transmits a beacon, having the structure described in figure. An ST first listens for a beacon and then sends a ranging request. The System S then sends a ranging response. The ranging response contains various connection IDs and other

information through which the subscriber terminal ST communicates the later steps with system S. After ranging, the ST enters the process of Registering with S. The detailed state transition diagrams are shown in Figure 3.4 and Figure 3.5

3.2.1.2 Registration

Through this procedure, the ST informs the System S that it is entering the set of ST serviced by S. The link between S and ST is connection-oriented: one or more connections can be established for data exchange. The registration process is required prior to any data connection formation. The process involves a registration request from the ST, followed by a registration response from S. During this process, ST and S exchanges operational parameters and capabilities. This process enables the ST to acquire IP address to setup provisioned connections.

The detailed state transition diagram is shown in Figure 3.6.

3.2.2 Connection Management

After registration, the ST can request for any number of further connections. The MAC is connection-oriented and data flow between BS and ST occurs as per the service flow type associated with that particular data flow. A new service can be added, or an existing service can be modified, or a service can be deleted. Thus, the connection management consists of Service Addition, Service Change, and Service Deletion.

3.2.2.1 Service Addition

In this phase, the entity (BS or ST) wishing to create a data connection exchanges a management message which installs a CID at BS and informs the destination about the nature of bandwidth request service to be used with the connection. The destination responds with a either acceptance or rejection of the request. Then the source sends an Acknowledgement. The detailed state transition diagrams are shown in Figure 3.8 and Figure 3.9.

3.2.2.2 Service Change

This may also be termed as the QoS Management phase. It is applicable to a new connection having a CID but not having any specified/allocated bandwidth resource. It is also applicable to an existing connection having some allocated resources but wanting a change in the allocation. In this phase, the entity (BS or ST) wishing to change characteristics of a data connection exchanges a management message to informs the peer entity. The peer entity responds with a response. Then the source responds with an acknowledgement. The detailed state transition diagrams are shown in Figure 3.10 and Figure 3.11.

3.2.2.3 Service Deletion

This may also be termed as the Connection Termination phase. In this phase, the entity (BS or ST) wishing to terminate a data connection exchanges a management message to informs the peer entity. The peer entity responds with a response. The detailed state transition diagrams are shown in Figure 3.13 and Figure 3.12

3.2.3 Bandwidth Request Grant Service

WifiRe provides following types of bandwidth request services:

3.2.3.1 Unsolicited Grant Service

The Unsolicited Grant Service (UGS) is designed to support real-time flows that generate fixed size data packets on a periodic basis, such as T1/E1 and Voice over IP without silence suppression. When a data CID is associated with UGS service flow type, the ST does not have to send periodic bandwidth request to the BS for that connection (data CID).

3.2.3.2 Real-time Polling Service.

The Real-Time Polling Service (rtPS) is designed to support real-time flows that generate variable size data packets on a periodic basis, such as MPEG video. The service offers real-time, periodic, unicast request opportunities, which meet the flow's real-time needs and allow the ST to specify the size of the desired grant.

3.2.3.3 Non real time Polling Service

The Non-Real-Time Polling Service (nrtPS) is designed to support non real-time flows that require variable size data grant slots on a regular basis, such as high bandwidth FTP. The service offers unicast polls on a regular basis, which assures that the flow receives request opportunities even during network congestion.

3.2.3.4 Best Effort Service

The intent of the Best Effort (BE) service is to provide efficient service to best effort traffic. Request/Transmission Policy setting should be such that the ST is allowed to use contention request opportunities.

3.2.4 Periodic Ranging

Following initial ranging, periodic ranging allows the ST to adjust transmission parameters so that the SS can maintain uplink communications with the BS. The detailed state transition diagram is shown in Figure 3.7

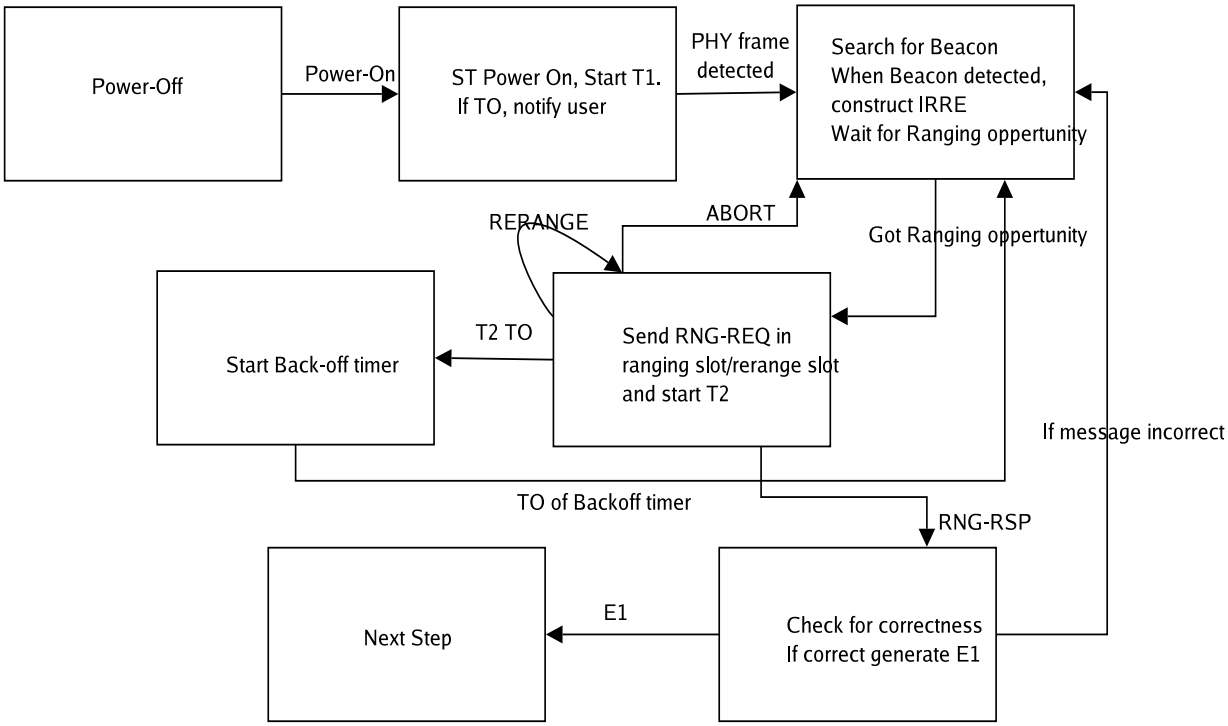

Figure 3.4: Initial Ranging ST

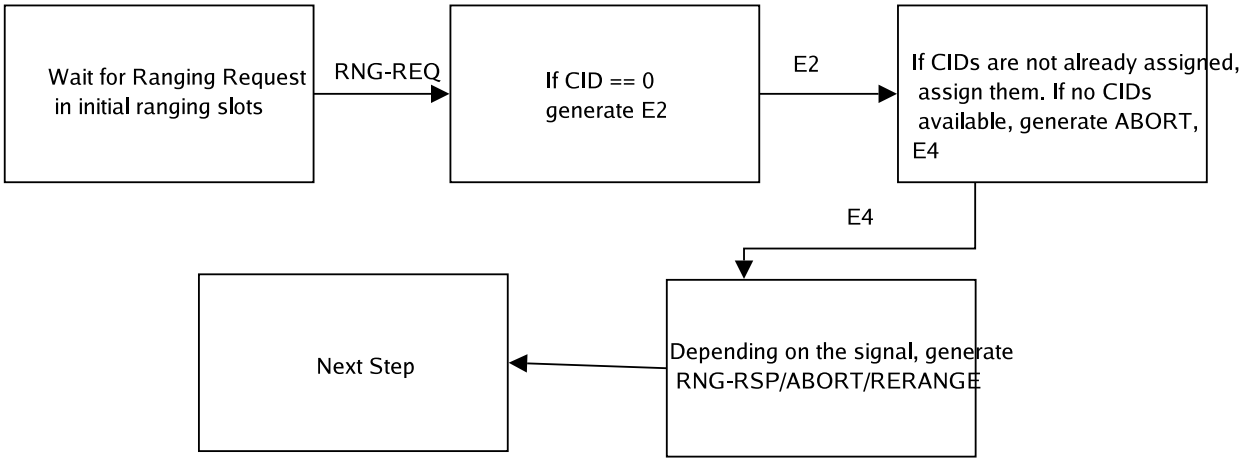

Figure 3.5: Initial Ranging BS

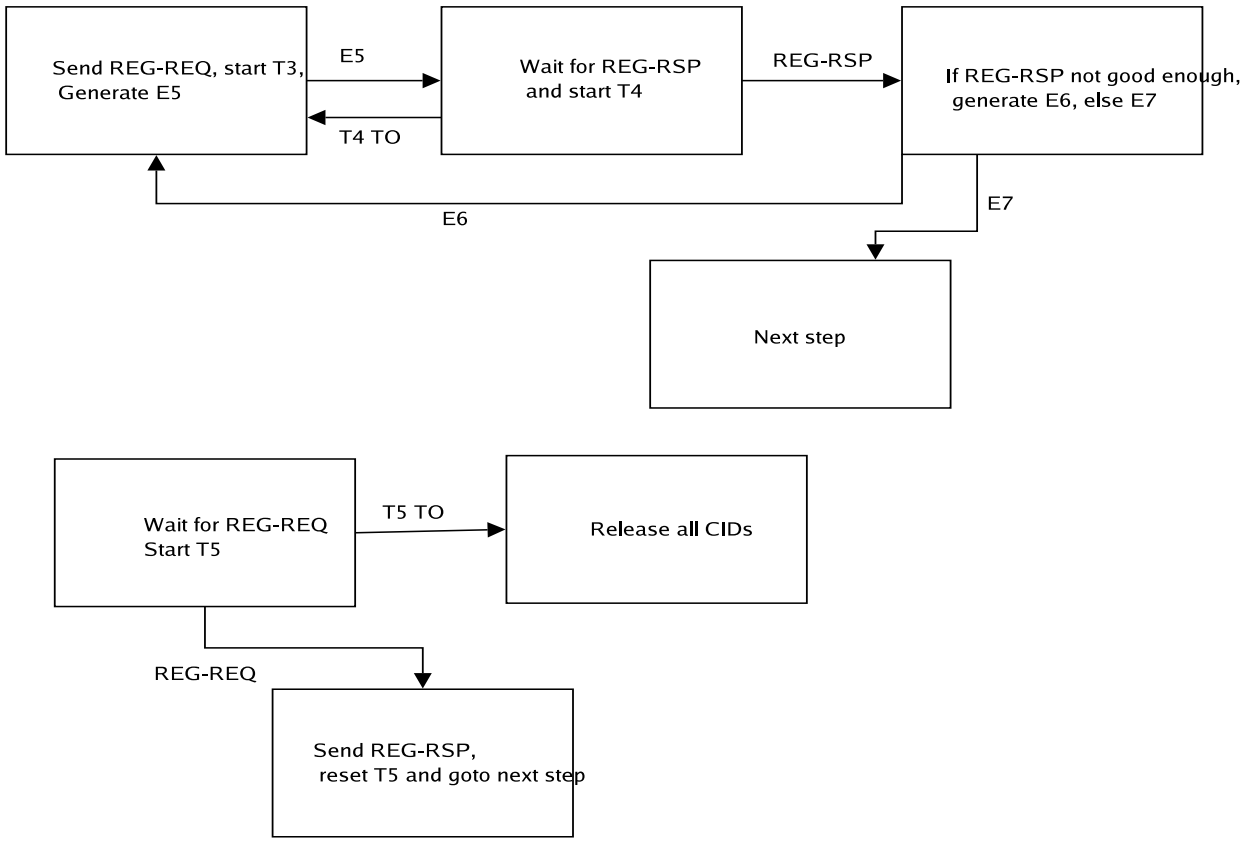

Figure 3.6: Registration ST and Registration BS

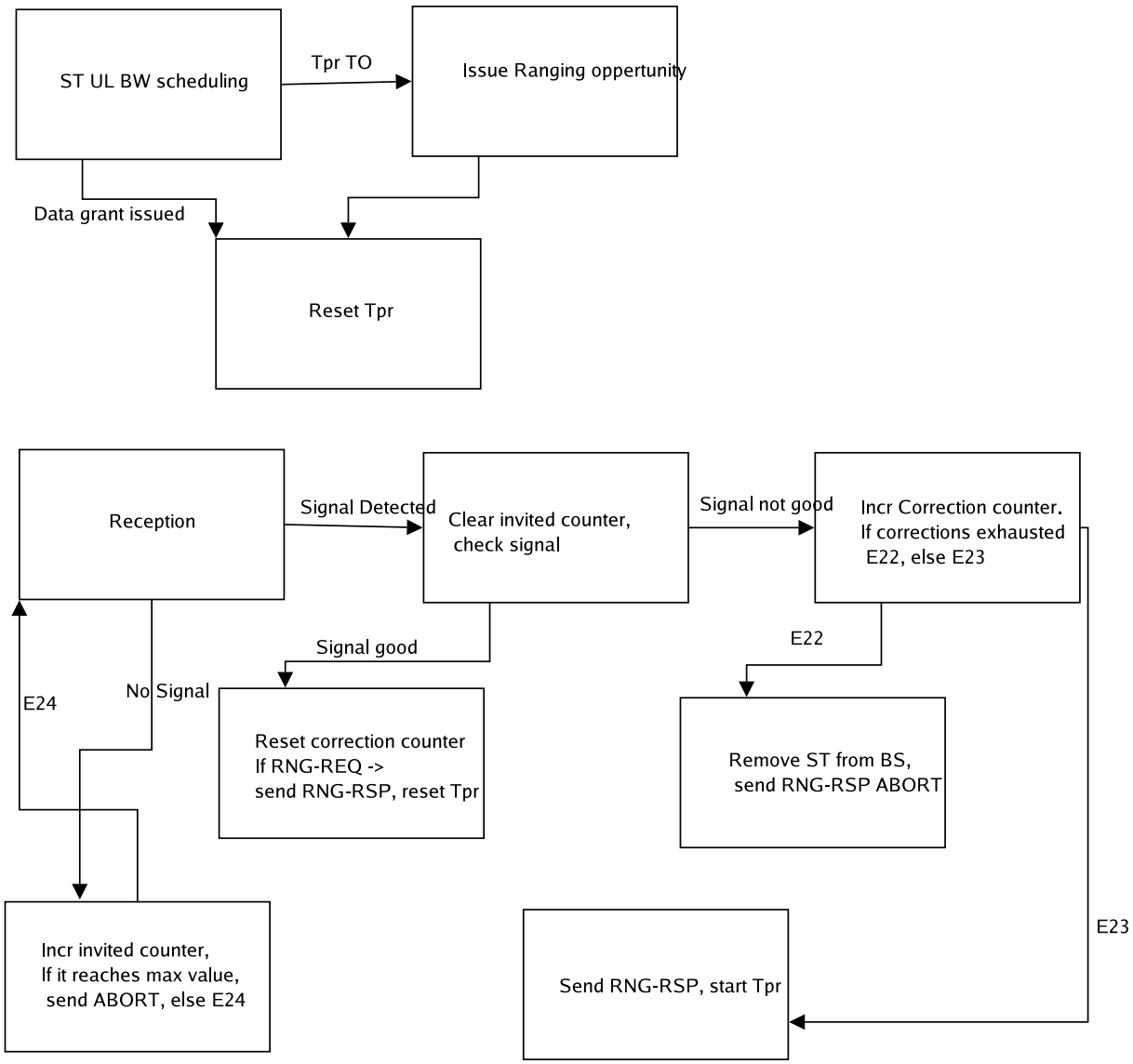

Figure 3.7: Periodic Ranging BS

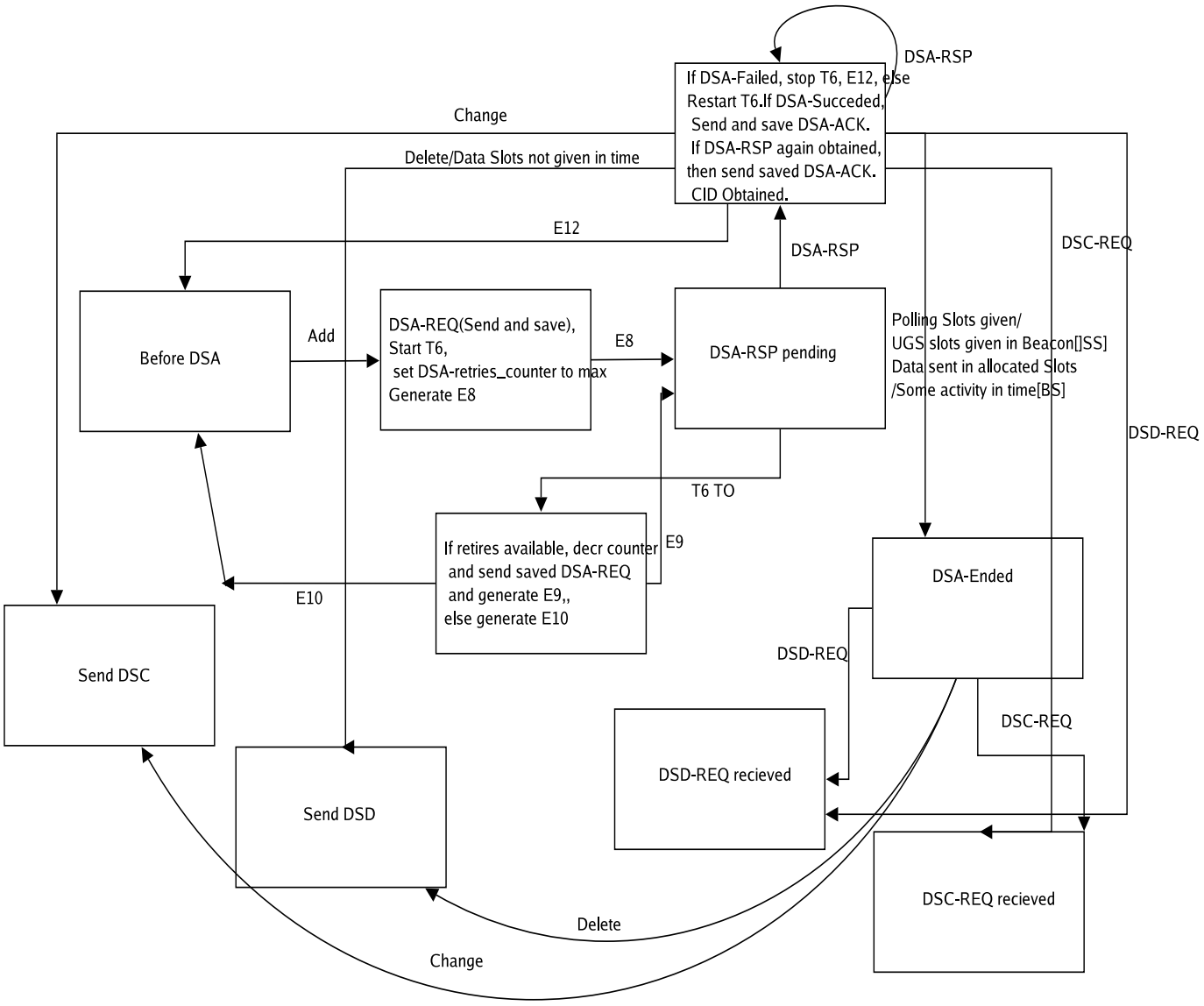

Figure 3.8: Locally initiated DSA

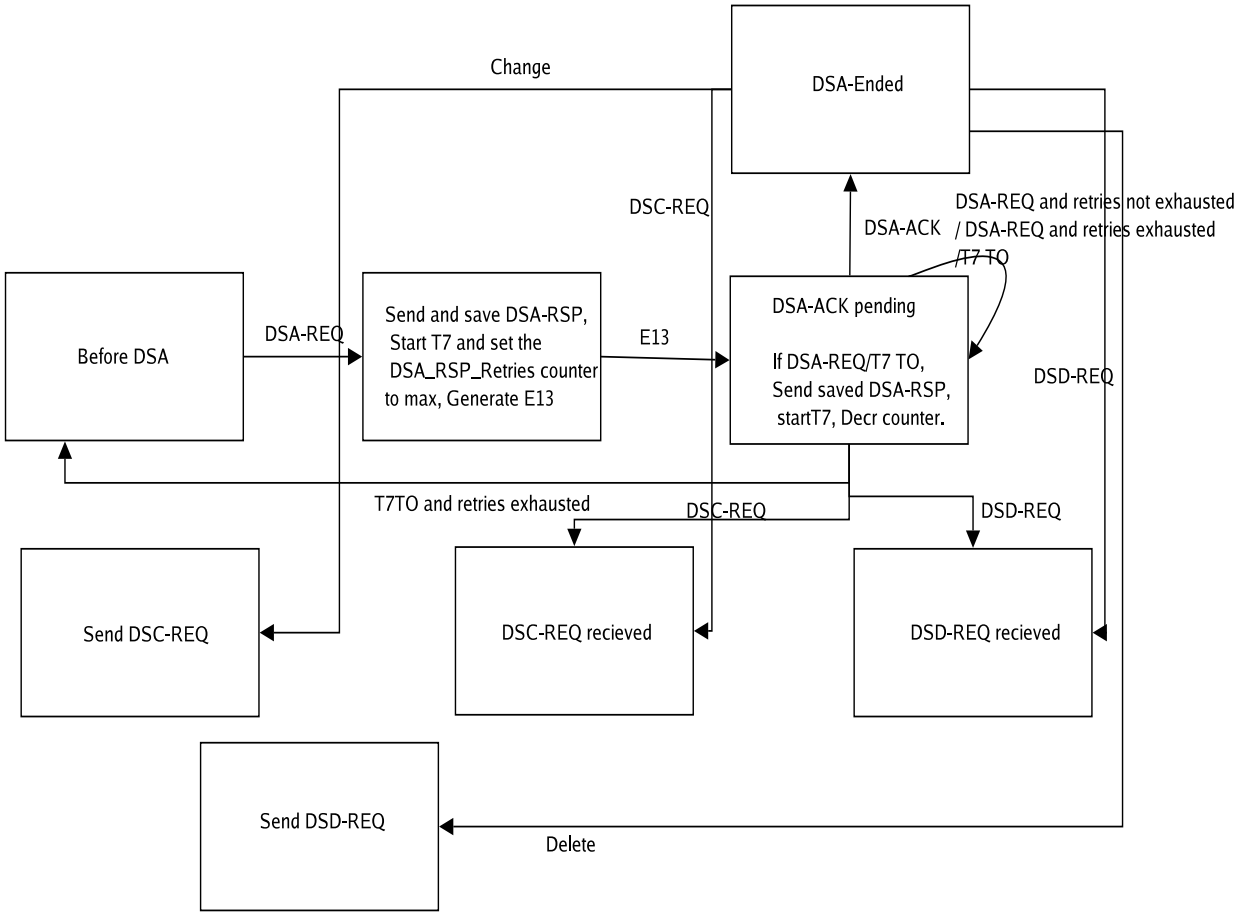

Figure 3.9: Remotely initiated DSA

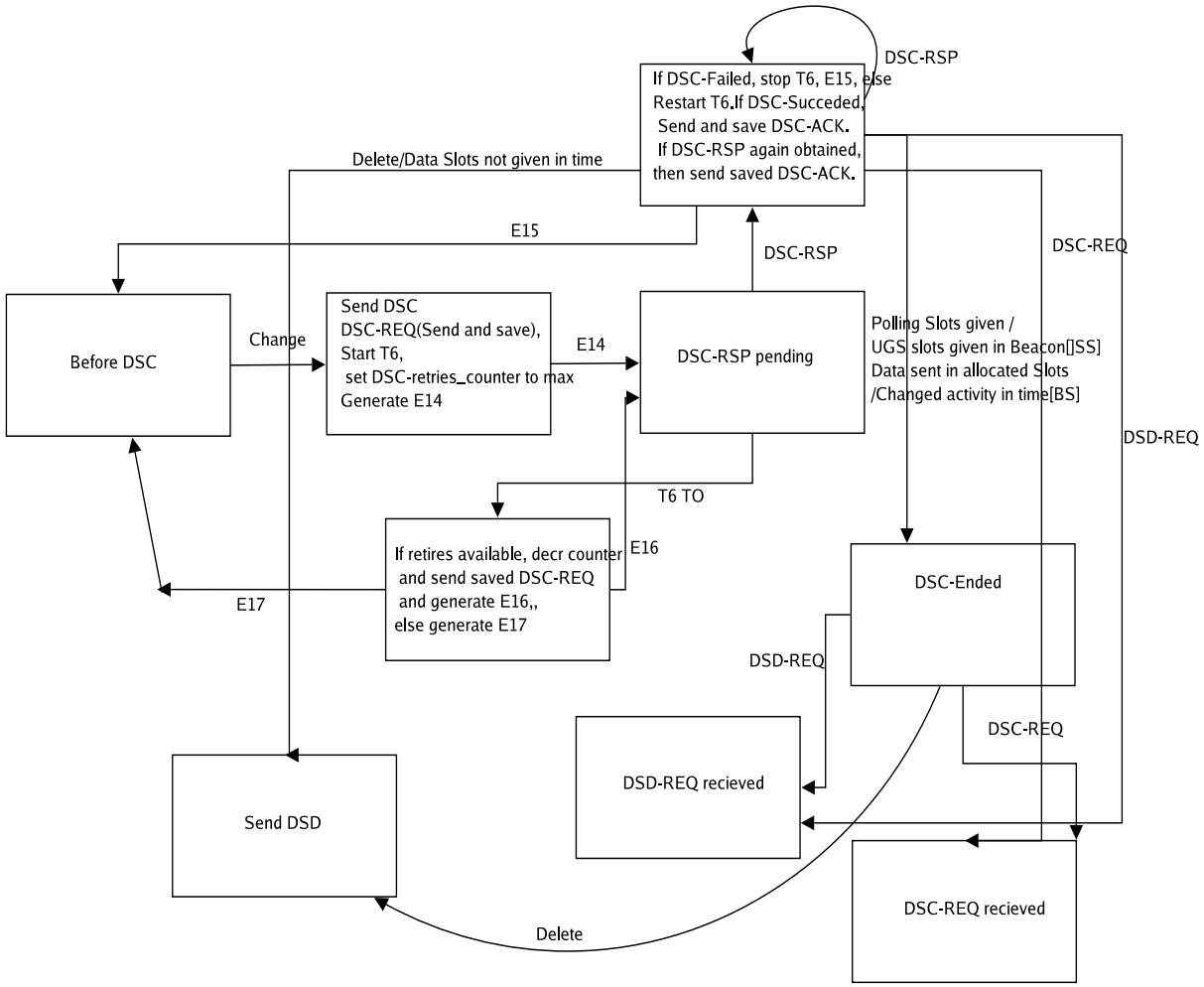

Figure 3.10: Locally initiated DSC

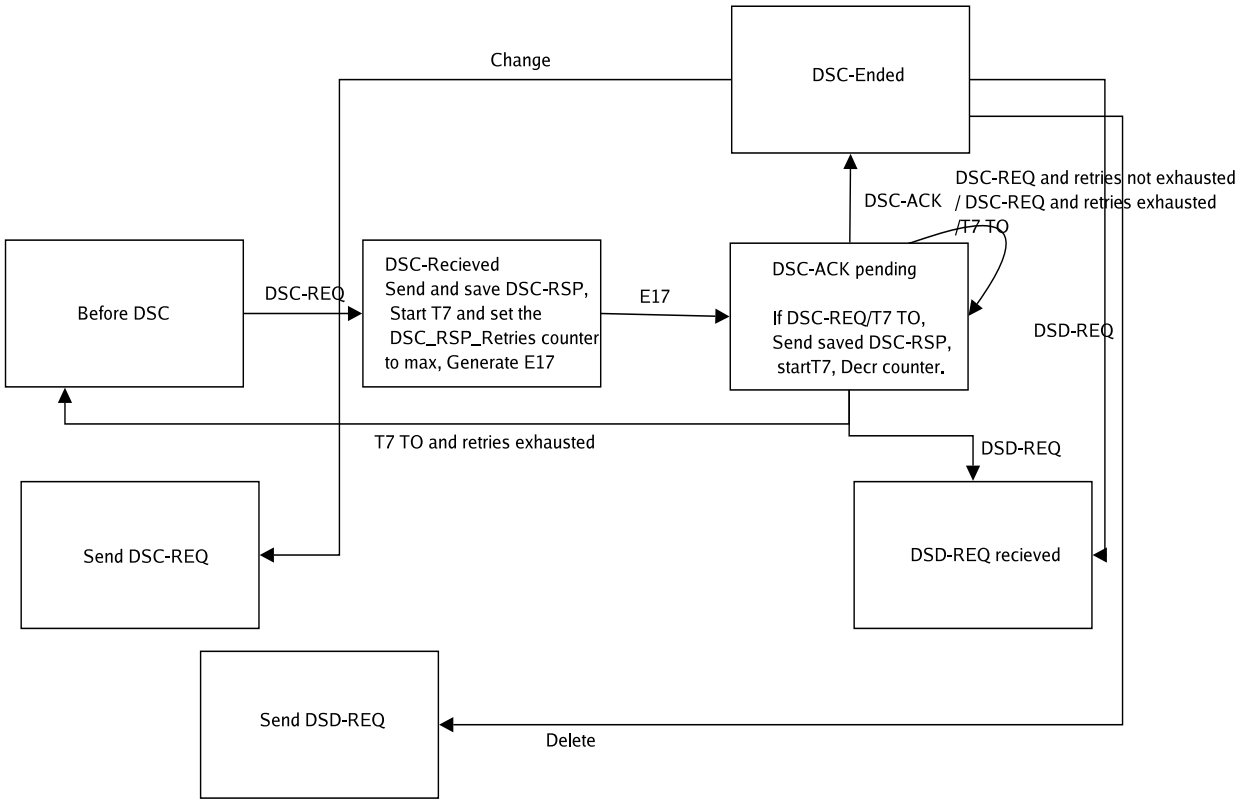

Figure 3.11: Remotely initiated DSC

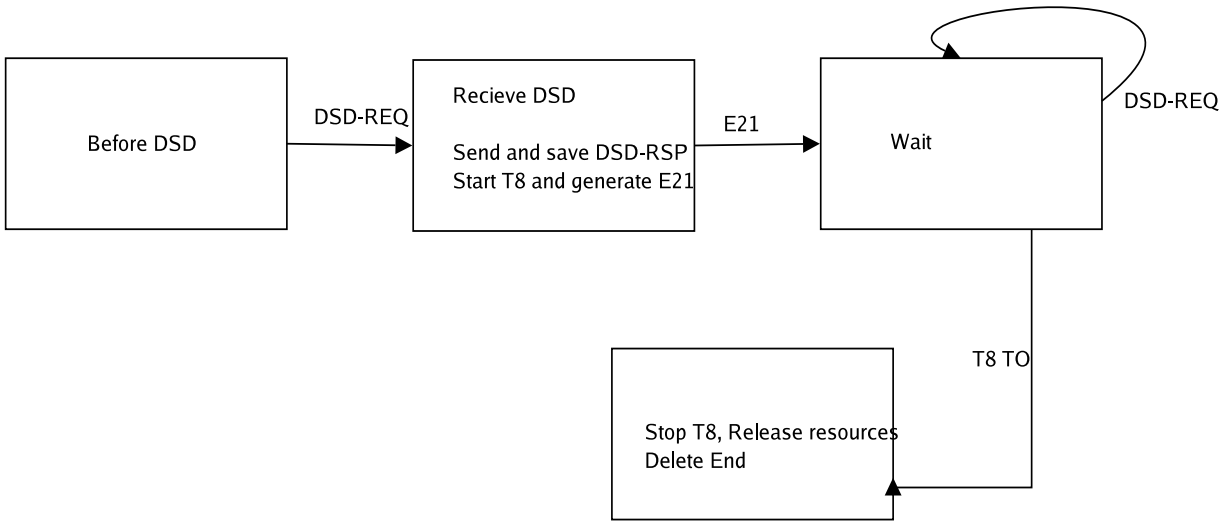

Figure 3.12: Remotely initiated DSD

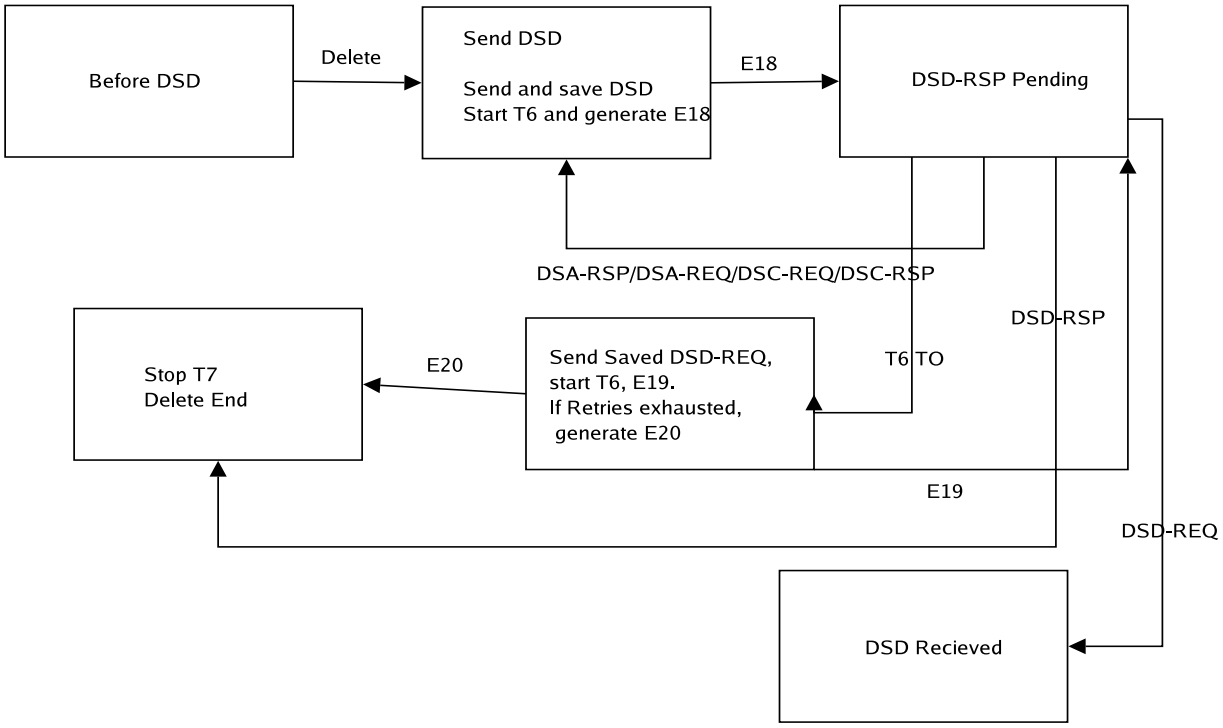

Figure 3.13: Locally initiated DSD

Chapter 4

Modelling of Protocol using Spin

PROMELA (PROcess MEta LAnguage) is the modeling language to describe concurrent (distributed) systems. SPIN (Simple Promela INterpreter) is a tool used for analyzing Promela programs leading to detection of errors in the design of systems. *e.g* deadlocks, race conditions, assertion violations, safety properties, liveliness properties etc.

We can use Spin in simulation mode or verification mode. Spin does not attempt to verify properties of a model directly, with any generic built-in code. It generates a verifier that can be compiled and run separately. This provides a significant gain in performance. The verifier is generated as a C program that is stored in a number of files with a fixed set of names, all starting with the three-letter prefix *pan*. These are *pan.h, pan.m, pan.b, pan.y, pan.c*. The file *pan.h* is a generic header file for the verifier that contains the translated declarations of all global variables, all channels and all process types. File *pan.m* defines the executability rules for all PROMELA statements used in the model, and the effect they have on the system state when successfully executed. File *pan.b* defines how the effect of each statement from *pan.m* can be undone when the direction of the depth-first search is reversed. File *pan.t* contains transition matrix that encodes the labeled transition systems for each process type. Finally *pan.c* contains the algorithms for computation, state space maintenance and cycle detection algorithms.

4.1 BS and ST Modeling

Each ST is modeled as a process. Each instance of process *ST()* is initiated in the begining. Each ST has as its parameters, various channels that are used for communication in the model. ST process is as shown below

```
proctype ST(unsigned i : 2; chan st_to_bs_0; chan bs_to_st_0;
chan timeout2_0;   chan timeout3_0; chan st_to_bs_reg_sim_0;
chan st_to_bs_1; chan bs_to_st_1;  chan timeout2_1;
chan timeout3_1; chan st_to_bs_reg_sim_1; chan st_to_bs_temp;
chan bs_to_st_temp; chan beacon_carrier)
```

The channels *st_to_bs*_0 and *st_to_bs*_1 are used for sending messages from ST to BS. Similarly *bs_to_st*_0 and *bs_to_st*_1 are used for communication from BS to ST. The channels *timeout*2_0, *timeout*2_1 and *timeout*3_0, *timeout*3_1 are used to model timeouts. The channel *beacon_carrier* is used for transmission of beacons from BS to ST. Channels *st_to_bs_temp* and *bs_to_st_temp* are used to model mutual exclusion between ST processes and BS process.

The BS is modelled as process *BS()*. The process prototype is as shoen below.

```
proctype BS(Connection connection_0; Connection connection_1;
Connection connection_2;  Connection connection_3; Connection1
connection1_0; Connection1 connection1_1)
```

Where *Connection* and *Connection*1 are structures which contains the channels required for communication between ST and BS. The instances of these structures are passed as parameters to this process.

4.2 Channel Modeling

Each connection has two channels one from ST to BS and other from BS to ST as shown. Each channel can contain at-most one message since in the protocol DL should be followed by UL.

```
chan bs_to_st = [1] of { mtype, byte };
chan st_to_bs = [1] of { mtype, byte, byte };
```

Promela provides the `timeout` statement to model system-wide timeouts: if no other statement in the global system is executable, the `timeout` statement becomes executable. But the global timeout has two disadvantages:

- The `timeout` statement cannot be used to check for erroneous deadlocks in the system; in such situations which should be recognized as deadlocks, the `timeout` statement will still be executable.
- Another problem is global nature of the `timeout`. There are situations where a 'local' timeout is demanded which becomes executable when only this process cannot proceed anymore. In this case, other processes in the system would not need to be stopped.

So we have gone for the following approach to model timeouts. Two timeout channels are defined which convey the message loss between BS and ST. These channels are as shown below.

```
chan timeout2 = [1] of { bit };
chan timeout3 = [1] of { bit };
```

timeout2 will be set whenever there is a loss of message that is sent from BS to ST, and *timeout3* will be set whenever there is a loss of message that is sent from ST to BS.

4.3 Modeling Broadcasting from BS to ST

There is no provision for broadcasting in Spin. We have modeled broadcasting by sending all messages to all STs. If a message is to be broadcasted it is sent on all the channels from BS to ST. We have broadcasted only beacon on all channels and individual messages we have sent to individual STs, because for messages other than beacons it does not make any difference even if all messages are sent to all STs or only the relevant messages are sent to the STs.

4.4 Modeling Alternative transmission of BS and STs

We have used two message channels to obtain mutual exclusion between STs and BS as shown below.

```
chan st_to_bs_temp = [1] of { bit };
chan bs_to_st_temp = [1] of { bit };
```

BS waits for message on all of these channels from ST to BS till its next transmission. ST also waits for message on channel from BS to ST until it's next transmission. These are shown below.
Waiting of ST for it's next transmission.

```
if
:: bs_to_st_temp?1   ->
      -
      -
      -
```

Waiting of BS for its next transmission.

```
atomic{
do
::((NSTindex<NSTs)&&(index_Array_Object[NSTindex].index_Array == NSTindex))->
if
:: (NSTindex == 0) ->
   st_to_bs_temp_0?_;
:: (NSTindex == 1) ->
   st_to_bs_temp_1?_;
fi;
-
-
```

4.5 Modelling Ranging Slots and Contention Slots

We have assumed only two ranging slots and two contention slots. An ST can send a message on any of the ranging slots(in case of ranging) or contention slots(in case of lost registration request message) at random. We have modeled this as follows.

Modeling of Ranging Slots: We have assumed that ranging slots have slot numbers 0 and 1. An ST can choose any of the ranging slots at random.

```
if
:: skip;
   slot_st = 0;

:: skip;
   slot_st = 1;
fi;
```

Modelling of Contention Slots: We have assumed that contention slots have slot numbers 10 and 11. An ST can choose any of the contention slots at random.

```
if
:: skip;
   slot_st = 10;

:: skip;
   slot_st = 11;
fi;
```

In the protocol if two or more STs transmit in the same ranging slot or contention slot, there should be retransmission since the BS cannot understand the message. In the model that we have implemented, the BS checks if there are two or more messages in the same ranging slot or contention slots. If there is more than one message, then these messages are discarded and a timeout signal is sent to the ST. In order to reduce the complexity of the protocol, we have assumed that once in every $Max\_Allowable\_No\_Of\_Transmissions\_On\_Contention\_Slots\_Without\_Success$ number of times, there is going to at least one successful transmission. Similarly in case of Ranging. Similar kind of assumption is taken for each slot.

4.6 Modeling Scheduler

We have implemented a random scheduler, which at random decides the number of connections it can permit before every downlink transmission. Depending on the priorities of the connection requests, the connections get accepted. If a connection gets deleted, changed, or a new connection gets added, the scheduler is called again and appropriate actions are performed depending on the number of connections now allowed by the scheduler. If a connection gets deleted, the scheduler increases the number of connections or allows for change of attributes of some connection depending on the priority. If a new connection request arrives, it may or may not get accepted. If a connection sends a request to change its attributes, the scheduler may permit new connections to get added, or permit change of connections or it may delete some connections. Taking a random decision about permitting a particular number of connections by following the priorities is valid because, in reality the scheduler takes the decision depending on the number of slots available and other dynamic data.

4.7 Optimization Options

During initial modeling of the protocol we had difficulty in managing the complexity of the verification runs. So we reduced the inherent complexity in the protocol. The following steps were followed.

- We removed all the redundant computations and redundant data using the output of SPIN option -A and through observation.
- We avoided using large range values, such as integer counters. We used the *unsigned* integer option in SPIN.
- We used channel assertions which helps in enhancing the performance of in-built partial order reduction algorithm in SPIN. If a channel-name appears in an xs xr channel assertion, messages may be sent to (received from) the corresponding channel by only the process that contains the assertion, and that process can only use send (receive) operations, or one of the predefined operators nempty or nfull. Also arrays of channels cannot be used. Because of these conditions, we could not avoid ladder code in our model.

```
xr st_to_bs__0;
xs bs_to_st__0;
xr st_to_bs_temp_0;
xs bs_to_st_temp_0;
xs beacon_carrier_0;
     -
     -
     -
```

- We reduced the number of slots in the channels.
- We used local variables instead of global variables. Local variables help partial order reduction algorithm in reducing the search space.
- We changed the local computations into atomic sequences. Atomic sequences reduce the verification complexity. We used atomic sections in the code as much as possible, as we noticed that the depth of the search and the size of the explored state space can be dramatically reduced this way.
- We avoided leaving scratch data around in local and global variables. When ever a variable is not being used or it is not used until next iteration, we reset it.
- We used special keyword *hidden* in the language. This keyword helps in hiding the variable from the verifier completely. If a variable is declared as hidden, then it does not add to the state space. It has been used to flush data out of channels.
- We used the predefined functions *nempty()* and *nfull()* instead of $len(q) > 0$ and $len(q) < MAX$. This helps the partial order reduction algorithm.
- Initially we had one process for each BS. But to avoid the complexity, we simulated multiple BSs in a single process.
- We have abstracted the protocol in order to reduce its complexity.

Chapter 5

Properties Checked

5.1 Unsatisfied Results

The following properties have been verified:

1. **Live-lock due to non deallocation of Basic CID and Primary CID:**

   The ST obtains basic CID and primary CID with the ranging response. But after some tine if the ST does not finish the registration, these CIDs need to be deallocated at the BS. But such deallocation of CIDs is not mentioned in the draft. This may cause a livelock scenario when all the CIDs are exhausted. The BS stops near the part where the allocation of CIDs is to made, and the ST also stops waiting for the CIDs. This scenario has been found in one of experiments conducted using Spin.

   The details of the experiment conducted are as shown below. To model this behavior we have assumed a simple scenario in which only single ST is present in the sector. The primary CID and basic CID are assumed to be allocated already. In such a scenario, the ST continuously sends ranging request and the BS continuously sends ABORT message since it ran out of connection IDs. So in such a case both ST and BS will be continuously sending messages which are of no productivity.
   In case where primary CID and basic CID are not allocated already the output is as follows:
   Output:

```
$spin -a 1.pl
```

```
$gcc -DBITSTATE -O2 -DNFAIR=4 pan.c
$./a.out -a -f -i -w29 -k10

State-vector 108 byte, depth reached 1810, errors: 0
421603 states, stored
480773 states, matched
902376 transitions (= stored+matched)
201280 atomic steps

hash factor: 1273.4 (best if > 100.)

bits set per state: 10 (-k10)
     -
     -
     -
```

In case where primary CID and basic CID are allocated already the output is as follows:

Output:

```
$spin -a 1.pl
$gcc -DBITSTATE -O2 -DNFAIR=4 pan.c
$./a.out -a -f -i -w29 -k10

pan: acceptance cycle (at depth 698)
pan: wrote 1.pl.trail
pan: reducing search depth to 1123
pan: wrote 1.pl.trail
pan: reducing search depth to 527
(Spin Version 4.2.7 -- 23 June 2006)
      + Partial Order Reduction

Bit statespace search for:
```

```
never claim             - (none specified)
assertion violations    +
acceptance   cycles     + (fairness enabled)
invalid end states      +

State-vector 108 byte, depth reached 1127, errors: 2
52547 states, stored
70656 states, matched
123203 transitions (= stored+matched)
46160 atomic steps

hash factor: 10217 (best if > 100.)

bits set per state: 10 (-k10)
     -
     -
     -
```

To overcome this problem we suggest a timer, before the expiration of which the BS should receive a Registration request form the ST which has completed ranging. If registration is not done before the timer expires, or if the BS due to some reason aborts its connection with ST, then these CIDs need to be deallocated.

2. **Every Ranging request must be followed by a Ranging Response:**

As suggested in the draft, if an ST sends a Ranging request, eventually it should be followed by a Ranging response. This property has been modelled as shown below. The LTL formula for the property is

$$[](RangingReq \longrightarrow <> RangingResp)$$

This is modelled as a never state in Promela as shown below:

Output:

```
never {    /* ![](stmt3 -> <>stmt4)
*/
T0_init:
     if
     :: (! ((stmt4)) && (stmt3)) -> goto accept_S4
     :: (1) -> goto T0_init
     fi;
accept_S4:
     if
     :: (! ((stmt4))) -> goto accept_S4
     fi;
}
Where stmt3 and stmt4 are defined as shown below:
#define stmt3 (connection[0].rng_Completed==0)
#define stmt4 (connection[0].rng_Completed==1)
```

This property is checked for single ST in a sector scenario. We have found that, the protocol goes into a live-lock if the BS continuosly sends ABORT message. To over come this we suggest a timer, the amount of time that the ST should stop transmitting so that there is a fair chance of the connection getting established. An alternative would be back-off mechanism. Part of the output is as shown below.

Output:

```
                    -
                    -
                    -
Beacon detected                        Abort signal sent i = 0
                else break; NSTindex = 1           Wait Until_Ulink
<<<<<START OF CYCLE>>>>>
                Beacon Send
                Before if              no_STs_in_Ranging_Slot[0] = 1
        Waiting for ranging req sent on ranging slot:  NSTindex = 0
                jus checkin else
```

```
                hey
                 INCR RNG on Ranging slot NSTindex = 1
                1st cond

Beacon detected                        Abort signal sent i = 0
                else break; NSTindex = 1          Wait Until_Ulink
spin: trail ends after 696 steps
                 -
                 -
```

3. **Every Registration request must be followed by a Registration Response:**

   The LTL formula for the property is

   $$[](Reg\_Req \longrightarrow <> Reg\_Resp)$$

   Similar to the Ranging process, if an ST sends a Registration Request, it should eventually receive a Registration Response. We have encountered the same problem as the previous case, and we propose the same solution to this problem.

4. Similar kind of problems were faced in case of DSA, DSC, DSD. So we suggest a timer in thoses cases as-well.

5.2 Satisfied Properties

1. The above properties were satisfied after placing a timer.

2. **Livelocks in Protocol till connection establishment:**

   The protocol is livelock free till connection establishment as shown.

   Output:

   -

   -

```
State-vector 236 byte, depth reached 2941, errors: 0
2.15329e+08 states, stored
2.25083e+08 states, matched
4.40412e+08 transitions (= stored+matched)
8.02195e+08 atomic steps

hash factor: 19.9461 (best if > 100.)

bits set per state: 3 (-k3)
        -
        -
```

3. **Potential Deadlocks in the Protocol till connection establishment:**

   We have checked the protocol under various scenarios for the presence of any potential deadlocks. We assumed two BSs and 2 STs. The 2 STs can be one for each BS or both STs can be in for one BS.

   Output:

```
$spin -a 1.pl
$gcc -DBITSTATE -DSC -O2 -DSAFETY pan.c
$./a.out -c0 -w28 -k3
        -
        -
State-vector 232 byte, depth reached 2359, errors: 0
5.09026e+07 states, stored
1.47327e+07 states, matched
6.56353e+07 transitions (= stored+matched)
1.48047e+08 atomic steps

hash factor: 84.3762 (best if > 100.)

bits set per state: 3 (-k3)
```

-

-

4. **Eventually Primary CID is deallocated:**

LTL property is as given.

$$[]((Primary\_CID == 1) \longrightarrow <> Primary\_CID == 0)$$

```
/*property Begins */ never {    /* !([](stmt1 -> <>stmt2)) */
T0_init:
     if
     :: (! ((stmt2)) && (stmt1)) -> goto accept_S4
     :: (1) -> goto T0_init
     fi;
accept_S4:
     if
     :: (! ((stmt2))) -> goto accept_S4
     fi;
}
```

Output:

-

-

```
State-vector 236 byte, depth reached 2915, errors: 0
1.137e+08 states, stored
1.12843e+08 states, matched
2.26543e+08 transitions (= stored+matched)
4.37733e+08 atomic steps

hash factor: 37.7745 (best if > 100.)
```

5. **Eventually Basic CID is deallocated:**

Similar to Primary CID, this property satisfied in case of Basic CID. LTL property is as given.

$$[]((Basic\_CID == 1) \longrightarrow <> Basic\_CID == 0)$$

```
\begin{verbatim}
          -
          -
  State-vector 236 byte, depth reached 2915, errors: 0
  1.137e+08 states, stored
  1.12843e+08 states, matched
  2.26543e+08 transitions (= stored+matched)
  4.37733e+08 atomic steps

  hash factor: 37.7745 (best if > 100.)
```

6. **Every** DSA_REQ **should be followed by a** DSA_RSP **or** DSA_ERRED**:**

   The LTL Formula for the same is as given

$$[]((dsa\_req\_Sent == 1) \longrightarrow <> (((dsa\_rsp\_obtained == 1)\&(dsa\_erred == 0))|((dsa\_rsp\_obtained == 0)\&(dsa\_erred == 1)))$$

   Output:

```
State-vector 432 byte, depth reached 3959, errors: 0
6.28167e+07 states, stored
3.5866e+07 states, matched
9.86826e+07 transitions (= stored+matched)
1.75818e+08 atomic steps

hash factor: 5.27332 (best if > 100.)
```

7. **Every** DSC_REQ **should be followed by a** DSC_RSP **or** DSC_ERRED**:**

The LTL Formula for the same is as given

$$[]((dsc\_req\_Sent == 1) \longrightarrow <> (((dsc\_rsp\_obtained == 1)\&(dsc\_erred == 0))|((dsc\_rsp\_obtained == 0)\&(dsc\_erred == 1)))$$

Output:

```
State-vector 432 byte, depth reached 4030, errors: 0
5.75512e+07 states, stored
1.75939e+07 states, matched
7.51451e+07 transitions (= stored+matched)
2.00183e+08 atomic steps

hash factor: 5.66429 (best if > 100.)
```

8. **After DSD_REQ is sent, eventually the connection is disabled from both sides:**

The LTL Formula for the same is as given

$$[]((DSD\_REQ == 1) \longrightarrow <> ((ST\_Over)\&\&(BS\_Over)))$$

Output:

```
State-vector 432 byte, depth reached 3917, errors: 0
6.18569e+07 states, stored
4.67217e+07 states, matched
1.08579e+08 transitions (= stored+matched)
2.35024e+08 atomic steps

hash factor: 5.33962 (best if > 100.)
```

9. **Every DSA_RSP should be followed by a DSA_ACK or DSA_ACK_ERRED:**

The LTL Formula for the same is as given

$$[]((dsa\_rsp\_sent == 1) \longrightarrow <> (((dsa\_ack\_obtained == 1)\&(dsa\_ack\_erred == 0))|((dsa\_ack\_obtained == 0)\&(dsa\_ack\_erred == 1)))$$

Output:

```
State-vector 432 byte, depth reached 3896, errors: 0
5.78167e+07 states, stored
3.9866e+07 states, matched
9.43726e+07 transitions (= stored+matched)
1.34818e+08 atomic steps

hash factor: 4.97332 (best if > 100.)
```

10. **Every DSC_RSP should be followed by a DSC_ACK or DSC_ACK_ERRED:**

The LTL Formula for the same is as given

$$[]((dsc\_rsp\_sent == 1) \longrightarrow <> (((dsc\_ack\_obtained == 1)\&(dsc\_ack\_erred == 0))|((dsc\_ack\_obtained == 0)\&(dsc\_ack\_erred == 1)))$$

Output:

```
State-vector 432 byte, depth reached 4026, errors: 0
5.94572e+07 states, stored
1.94639e+07 states, matched
8.11451e+07 transitions (= stored+matched)
2.23467e+08 atomic steps

hash factor: 5.46429 (best if > 100.)
```

11. **Every connection terminates eventually:**

The LTL Formula for the same is as given

$$[](connection\_established == 1) \longrightarrow <> (connection\_established == 0))$$

Output:

```
State-vector 432 byte, depth reached 3912, errors: 0
6.18345e+07 states, stored
4.634217e+07 states, matched
1.08579e+08 transitions (= stored+matched)
2.24502e+08 atomic steps

hash factor: 5.34962 (best if > 100.)
```

12. **DSC should be after DSA:**

The LTL Formula for the same is as given

$$[](dsc\_start == 0)|((dsc\_start == 0) \bigcup (dsa\_Over == 1))$$

Output:

```
State-vector 236 byte, depth reached 3656, errors: 0
hash factor: 9.76 (best if > 100.)
```

13. **DSD should be after DSA:**

The LTL Formula for the same is as given

$$[](dsd\_start == 0)|((dsd\_start == 0) \bigcup (dsa\_Over == 1))$$

This property is satisfied similar to the above property.

14. **Data CIDs should be deallocated after connection termination:**

The LTL Formula for the same is as given

$$[]((Data\_CID == 1) \longrightarrow <> Data\_CID == 0)$$

If a $Data\_CID$ is allocated, eventually it is to be deallocated. **Output:**

```
State-vector 432 byte, depth reached 4094, errors: 0
5.98345e+07 states, stored
4.734217e+07 states, matched
1.28579e+08 transitions (= stored+matched)
2.567802e+08 atomic steps

hash factor: 5.2312 (best if > 100.)
```

15. **UGS should precede rtPS, nrtPS, BE traffic:**

This property has been proved using assertion statements in the admission control module. The admission control module first schedules all UGS connections followed by rtPS, nrtPS and BE traffic. While scheduling non UGS connections, the admission control module checks if there are any UGS connections which are eligible but are not allocated bandwidth. The assertion statement is as follows:

$$assert((number\_Of\_Eligible\_Connections[0] == 0))$$

Where $type_Index$ represents the present scheduling type of service in the scheduling module. There are a total of 4 types of services. Part of code is as provided.

```
do
:: (type_Index < 4) ->
   /*Admission control module. First it admits all
     connections of type = 0, then type = 1, ....*/
        -
        -
if
```

```
 :: type_Index == 1 ->
assert((number_Of_Eligible_Connections[0]==0));
        -
        -
        -
:: (type_Index == 4) ->
   break;

od;
```

And $number\_Of\_Eligible\_Connections$ is an array of size 4 (Number of types of service). Each element contains, the number of eligible connections of that particular type.

Output:

```
State-vector 428 byte, depth reached 3126, errors: 0
5.70195e+07 states, stored
1.05099e+07 states, matched
6.75294e+07 transitions (= stored+matched)
1.81262e+08 atomic steps

hash factor: 4.90778 (best if > 100.)
```

16. **rtPS should precede nrtPS, BE:**

Similar to the above property this property has been proved using assert statements. The assert statement is as follows:

$$assert((number\_Of\_Eligible\_Connections[1] == 0))$$

```
State-vector 428 byte, depth reached 3126, errors: 0
  5.70195e+07 states, stored
  1.05099e+07 states, matched
  6.75294e+07 transitions (= stored+matched)
```

```
1.81262e+08 atomic steps

hash factor: 4.90778 (best if > 100.)
```

17. **nrtPS should preced BE:**

Similar to the UGS property, this property has been proved using assert statements. The assert statement is as follows:

$$assert((number\_Of\_Eligible\_Connections[2] == 0))$$

```
State-vector 428 byte, depth reached 3126, errors: 0
  5.70195e+07 states, stored
  1.05099e+07 states, matched
  6.75294e+07 transitions (= stored+matched)
  1.81262e+08 atomic steps

  hash factor: 4.90778 (best if > 100.)
```

18. **The protocol is Deadlock free in connection Management phase:**

The protocol has been proved to be deadlock free in connection management phase.

```
  State-vector 416 byte, depth reached 3129, errors: 0
  7.9363e+08 states, stored
  1.36263e+08 states, matched
  9.29893e+08 transitions (= stored+matched)
  2.61501e+09 atomic steps

  hash factor: 5.4118 (best if > 100.)
```

19. **The protocol is Livelock free in connection Management phase:**

The protocol has also been proved to be livelock free in connection management phase.

```
State-vector 424 byte, depth reached 3895, errors: 0
5.93358e+07 states, stored
3.16119e+07 states, matched
9.09477e+07 transitions (= stored+matched)
1.9369e+08 atomic steps

hash factor: 4.924 (best if > 100.)
```

20. **Other Properties:**

In addition to the above properties, we also proved some other properties. These include, prperties like Registration should be after ranging, DSA/DSC/DSD phase should be after registration, DSC request from BS should be given priority in comparision to DSC from ST.

5.3 Results Accuracy

Initially we tried to obtain the results by searching for complete state space. The search was incomplete when we used no run-time optimizations. The main problem was the memory consumption. Then we tried to obtain results using hash-compact technique, but the search was not complete using even this option. So we obtained the results using bit-state hashing optimization technique. In the begining, even after using the bit-state hashing, we could not get proper accuracy. The main issue of the SPIN toolset is obviously related to scalability; the verification engine could not scale well with the more complicated system (more STs and BSs). Apparently, trying to incorporate the more number of components into the system (even using most coding optimizations) significantly increased the complexity of the system model. So we have to abstract the model in order to obtain results with higher accuracy.

All the experiments for connection establishment phase were done under 2 STs and 2 BSs assumption. The 2 STs can be part of any of the BSs. The BSs can be in the same

sector or they can be from different sectors. All experiments for connection management phase were sone under 2STs and 1 BS assumption.

All verification results mentioned in the previous section took less than 60 minutes to verify after using sufficient optimization techniques. All the results were obtained on a computer having 1 GB memory. The main problem was the memory consumption. In almost all the cases search went off limits over 10^8. An indication of the coverage of a search when verified using bit-state hashing technique can be derived from the hash factor. SPIN uses *hash-factor* to serve as a predictor function, which is defined as : $Hf = M/N'$ Where N' is the number of states that was reached and M is the total number of bits in memory that is available for performing the reachability analysis.

A hash-factor above 100 is nearly 100% accurate, as the hash-factor reaches 1, we cannot trust the results of the verification. Any hash-factor above 10 means around 98% accurate, a hash-factor of around 4 is around 93% accurate. Bit-state hashing can be performed using different number of hash-functions. The default optimal number of hash-functions is supposed to be 2 [14].

Chapter 6

Conclusion and Future Work

This report is about the application of PROMELA language and SPIN tool to verify a practical implementation of WiFiRe protocol. In this report we describe the setting of model and correctness properties and some problems in the protocol. We came across interesting problems regarding the reducing the complexity of the system and were forced to solve these problems with non-trivial effort. Even though we did not manage to verify the LTL properties for more complicated examples, we proved some useful properties of the protocol. The following are the major findings in the protocol.

- After power-on, if the ST does not see any beacon for a long time, it should report the user about the same. For this we need a timer and a counter. If the ST does not receive a beacon and the timer expires, it should try again. It should try for a specified number of times(counter value), and if it does not receive the beacon, it should report the user that the BS may not in power-on mode.

- We have suggested an Abort message, and Rerange message in addition to the Ranging Response. If there are not enough active STs in the sector, then the BS can send a slot(Rerange) asking the ST to transmit its ranging request on it. Also if there are many STs in the sector and an ST goes on sending Ranging request and loosing either the request itself or the response, then the BS can send an Abort message saying it to start all over again. This message can also be useful in periodic ranging and during data transmission phase.

- The ST obtains basic CID and primary CID with the ranging response. But after some time if the ST does not finish the registration these CIDs need to be deallocated at the BS. But such deallocation of CIDs is not mentioned in the draft. This may cause a livelock scenario when all the CIDs are exhausted. The BS stops near the

part where the allocation of CIDs is to made, and the ST also stops waiting for the CIDs. This scenario has been found in one of experiments conducted using Spin. To overcome this problem we need a timer, before the expiration of which the BS should receive a Registration request form the ST which has completed ranging.

- In the specification of WiFiRe, there is no acknowledgment message in return to DSA response. This may cause unnecessary wastage of slots as in the following scenario.

  If ST sends DSA request to BS and DSA response from BS is lost repeatedly. BS assumes that there is a connection established between ST and BS. It keeps on allocating polling slots to ST on that particular CID, until it finds out that the connection was not established. Also it may try in vain to send DSC request or DSD request to the ST on that particular CID until it finds out that connection is not established.

- There is no provision for acknowledgment message in return to DSC response. This may cause problems similar to those in case of DSA request. Consider the following scenario.

  ST sends DSC request and BS responds with a DSC response, but it is lost repeatedly. BS assumes that CID has been changed, and ST assumes that CID has not been changed. Eventually they will be wasting slots until they find out that the connection attributes are not changed.

- The properties mentioned in chapter 5 have been proved to be satisfied.

- We have come up with detailed state transition diagrams for various phases in the protocol.

The possible future work includes modelling proper scheduler, and incorporating data transmission with MAC level ARQ. This may require reducing the complexity of protocol so that some complex LTL properties can be checked with good accuracy.

Bibliography

[1] Anurag Kumar Sridhar Iyer, Krishna Paul and Bhaskar Ramamurthy. Wifire: Medium access control (mac) and physical layer (phy) specifications, May 2006.

[2] LAN/MAN standards Committee. Part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. IEEE-SA Standards Board, june 2003.

[3] LAN/MAN standards Committee, IEEE Microwave Theory, and Techniques Society. Air interface for fixed broadband wireless access systems. IEEE-SA Standards Board, june 2004.

[4] Spin Documentation. http://spinroot.com/spin/whatispin.html.

[5] Bhaskaran Raman Pravin Bhagavat and Dheeraj Sanghi. Turning 802.11 inside-out. ACM SIGCOMM, Jan 2004.

[6] Gerard J. Holzmann. *The SPIN model checker: Primer and reference manual*. Addison Wesley, 2004.

[7] Murphi Documentation. http://verify.stanford.edu/dill/murphi.html.

[8] Y. Dong, X. Du, Y. S. Ramakrishna, C. R. Ramakrishnan, I.V. Ramakrishnan, S. A. Smolka, O. Sokolsky, E. W. Stark, and D. S. Warren. Fighting livelock in the i-Protocol: A comparative study of verification tools.

[9] Gerard J. Holzmann. The engineering of a model checker: The gnu i-protocol case study revisited. In *Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking*, pages 232–244, London, UK, 1999. Springer-Verlag.

[10] Benchmarks for Explicit Model checkers Database. http://anna.fi.muni.cz/models/index.html.

[11] T. Ruys and R. Langerak. Validation of bosch' mobile communication network architecture with spin, 1997.

[12] Karthikeyan Bhargavan, Davor Obradovic, and Carl A. Gunter. Formal verification of standards for distance vector routing protocols. *J. ACM*, 49(4):538–576, 2002.

[13] Sohel Khan Syed M.S. Islam, Mohammed H. Sqalli. Modeling and formal verification of dhcp using spin. International Journal of Computer Science and Applications, june 2006.

[14] Gerard J. Holzmann. An analysis of bitstate hashing. Journal Formal Methods in System Design.