Policy Reasoning for Spectrum Agile Radios

Amol A. Deshpande

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Jung-Min "Jerry" Park, Chair
Michael S. Hsiao
Tamal Bose

May 4, 2010
Blacksburg, Virginia

Keywords: Policy Reasoning, Cognitive Radios, Spectrum Access Policies

Policy Reasoning for Spectrum Agile Radios

Amol A. Deshpande

ABSTRACT

DARPA's neXt Generation (XG) communication program proposes the use of Dynamic Spectrum Access (DSA) wherein intelligent radios can realize opportunistic usage of frequency bands by identifying the under-utilized spectrum and reasoning about it. Implementing such a flexible scheme requires changes in the current static spectrum management approach. As a result, declarative spectrum management through policy-based dynamic spectrum access has garnered significant attention recently.

Policy-based dynamic spectrum access decouples the *Spectrum Access Policies* and *Policy Processing Components* from the *Radio Platform*. The Policies define conditions under which the radios are allowed to transmit in terms of frequencies used, geographic locations, time etc. The Policy Processing Components include a reasoning engine called the *Policy Reasoner*, which is responsible for enforcing these policies.

This thesis describes the design and implementation of a novel policy reasoner called *Binary Decision Diagram based Reasoner for processing Spectrum Access Policies* (*BRESAP*). BRESAP processes spectrum policies efficiently by reframing the policy reasoning problem as a graph based Boolean function manipulation problem. BRESAP uses Binary Decision Diagrams (BDDs) to represent, analyze and process the policies. It uses a set of efficient graph-theoretic algorithms to merge these policies into a single meta-policy and compute

opportunity constraints.

Our policy reasoner has the capability to respond to invalid and under-specified transmission requests sent by the System Strategy Reasoner (SSR). In case of invalid or under-specified transmission requests, BRESAP returns a set of opportunity constraints which inform the SSR of the changes needed to the transmission parameters in order to make them conform to the policies. We also propose three algorithms for computing the opportunity constraints. The complexity of the first algorithm is proportional to the number of variables in the meta-policy BDD, while the complexities of the second and third algorithms are proportional to sum of number of variables and the size (i.e., number of nodes) of the meta-policy BDD.

I dedicate this thesis to

Aai, Baba and Prachi

for their constant love, support and encouragement

Acknowledgments

I would like to express my gratitude for my family who has been very supportive of my studies and graduate education. I could not have completed this work without the love and faith of my parents - Pradnya Deshpande and Anant Deshpande and my sister - Prachi.

I would like to sincerely thank my advisor Dr. Jung Min "Jerry" Park for his continued guidance and insightful suggestions throughout the duration of my research. Thank you for your faith in me and I am extremely privileged to be a student under your guidance.

I am also thankful to Dr. Hsiao and Dr. Bose for agreeing to be on my committee.

I would also like to thank my friends in ARIAS - Swati Kanaujia, Behnam Bahrak, Jatin Thakkar, Maxwell Whitekar, Kaigui Bian, and Daniel Ali for enriching my research experience and for their constructive suggestions and timely help. You guys are amazing. Behnam and Max, I enjoyed every minute of working with you and have learned a lot from just observing the two of you! I had a great time this past year and I wish you all the best of luck for the future.

I would also like to thank my friend Nikhil Rahagude whose suggestions went a long way in

helping me to write this thesis.

I would also like to take this opportunity to thank my roommates - Avinash Laxminarayana, Mahesh Nanjundappa and Nagendra Krishnamurthy who have made the past two years unforgettable. Life here just wouldn't be the same without you guys!

Finally, I am thankful to God for His countless blessings.

Grant Information

This work was partially sponsored by the National Science Foundation through grants CNS-0627436, CNS-0746925, and CNS-0910531. Any opinions, results and conclusions or recommendations expressed in this material and related work are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Spectrum Agile Radios with the capability of dynamic spectrum access can satisfy the rapidly increasing needs of spectrum users through efficient usage of spectrum opportunities. However, in order to realize the full potential of Spectrum Agile Radios, a spectrum access control mechanism is required that not only ensures its flexibility of reconfiguration but also ensures the flexibility of regulating radio behaviors in different domains. Recently, the policy-based management approach has been proposed to resolve the flexibility concerns in regards to the spectrum regulations [5] [6].

The policy based approach decouples policy enforcement from device implementation and optimizations. This decoupling of the policies and the radio platform has a number of

important advantages [7].

- In legacy radios policies are hardwired into the radio and are a part of the Radio Firmware. However, this approach has obvious drawbacks. In the policy-based spectrum sharing radios, policies can be dynamically loaded onto the radio without the need of recompiling any software on the radio. This brings in, the concept of *Policy Agility*. Policy Agility guarantees that the regulatory policies can be changed on the fly with *time* and *geography*. This leads to radios that are more flexible.

- Decoupling policy enforcement from device implementation also reduces the accreditation efforts. This is because the policies, policy processing components and the devices can be accredited and certified independently of each other. If any one of these components changes, only that particular component needs to be accredited without accrediting the entire system.

- Today there exists a dependency between the regulatory bodies and the technology wherein the regulatory bodies wait for technology to mature and technology needs to wait to see how the regulatory bodies specify usage for the policies. Decoupling the devices and the policies allows the two to evolve independently of each other.

- The policy-based approach is extensible with respect to the kinds of policies that can be expressed. An unprecedented amount of freedom and control of spectrum is possible as stakeholders can formulate spectrum policies (allowed by regulations) to best fit their needs.

From the above points we can see that decoupling the policies is not just beneficial but very much a necessity for implementing Dynamic Spectrum Access and Spectrum Agile Radios. Each Spectrum Agile Radio is equipped with a set of *Policy Conformance Components* (*PCCs*), which are responsible for ensuring that the radio's behavior conforms to the currently active set of policies and does not cause harmful interference [6]. The main policy enforcement component in a spectrum agile radio is a reasoning engine called the *Policy Reasoner*(*PR*). The policy reasoner carries out a number of tasks related to policy analysis and processing and assists in policy enforcement by evaluating transmission requests.

The core of the Policy Reasoning problem is the following: given a set of policies and a transmission request, the reasoner needs to determine whether the transmission request is allowed or denied in the context of the currently active policies. If the transmission request is denied, the policy reasoner needs to return a set of opportunity constraints which when satisfied allow transmission.

Under the auspices of DARPA's XG program, SRI International [4] [7] [11] [26] and Shared Spectrum Company [5] [6] have been developing policy-based spectrum access control systems. The field tests and demonstrations have illustrated that the policy reasoner can incur large memory overhead and processing delay unless its design and implementations are optimized. In light of the complexity of the policies and the dynamic nature of spectrum availability, it is evident that more research is needed in the design and development policy reasoners.

1.2 Contribution and Related Publications

We propose a new Policy Reasoner called *BRESAP* (*Binary decision diagram-based REasoner for Spectrum Access Policies*). BRESAP evaluates the transmission request sent by the *System Strategy Reasoner* (*SSR*) in reference to the currently active set of policies and sends back a reply which is either "Allow Transmission", "Deny Transmission" or "Transmission Request is Incomplete". In case the transmission request is denied or is incomplete, BRESAP also returns a set of *Opportunity Constraints* which helps the SSR understand the reason for denial of the transmission request and the changes that need to be made in the transmission request parameters so that the transmission is allowed.

BRESAP uses a set of efficient graph theoretic algorithms to translate policies into *BDDs*. We show that using BDDs and MTBDDs, the policies can be easily combined into a single *meta-policy* for the purpose of reasoning and returning opportunity constraints. Thus, essentially we reframe the reasoning problem as a graph based Boolean function manipulation problem.

We introduce the concept of *cost* for the radio parameters based on the difficulty level of the radio to change the values of these parameters. We present three graph-based reasoning algorithms for returning the constraints. The first algorithm ignores the costs while computing the opportunity constraints. The other two algorithms are based on weighted graph traversal and return constraints for the *Optimal Opportunity Constraints*.

We also propose another reasoner whose policy combining approach is based only on *BDDs* instead of MTBDDs. This reasoner has reduced memory requirements as compared to the

MTBDD-based reasoner.

The research contributions of this thesis can be summarized as follows:

1. Implementation of BRESAP.

2. Design and implementation of the BDD-based reasoner.

3. Design of the policy parser module in BRESAP - The policy parser module processes the Semantic Web Rule Language First Order Logic (SWRL FOL) [33] policies. This module uses the JDOM [34] Java API and the knowledge of the SWRL FOL language constructs to convert the policies into Boolean expressions.

4. Design of the policy preprocessor module in BRESAP - The preprocessor module semantically analyzes the active policies to generate the environmental constraints for dynamic spectrum access. These constraints ensure that the policy reasoner does not return illogical opportunity constraints to the SSR.

5. Design of the AllMinCostPath algorithm used by the Reasoner module in BRESAP. This algorithm returns all the optimal opportunity constraints to the SSR.

Some of the work presented in this thesis were published in the following conference papers:

1. B. Bahrak, A. Deshpande, M. Whitaker, and J. Park, "BRESAP: A policy reasoner for processing spectrum access policies represented by binary decision diagrams", IEEE DySPAN, Apr. 2010.

2. B. Bahrak, A. Deshpande, and J. Park, "A policy reasoner for policy-based dynamic spectrum access", SDR '09 Technical Conference and Product Exposition, Dec. 2009.

1.3 Thesis Organization

Chapter 2 starts with a brief overview of Dynamic Spectrum Access. This is followed by explanation of the XG radio architecture along with the policies used in the XG radios. Next follows a brief description of the different policy representation methods, BDDs and MTBDDs. A detailed analysis of the research currently going on, on the policy reasoner wraps up this chapter.

Chapter 3 gives a detailed block by block explanation of BRESAP architecture and the reasoning algorithms used by the reasoner. It also explains the implementation of a memory efficient BDD based policy reasoner.

Chapter 4 gives the results of experimental evaluation of the reasoner and then wraps up the dissertation with the conclusion.

Chapter 2

Technical Background

2.1 Dynamic Spectrum Access

Spectrum is a finite and limited resource. In the current static spectrum allocation scheme, every new service is allotted a fixed block of spectrum. Consequently, we are slowly nearing the time when there will be no more spectrum left for assigning.

Also today different regions or countries have different spectrum environments and different spectrum governing policies. As a result of this, extensive coordination is required amongst the authorities of these regions/countries to guarantee issueless operation which leads to large deployment delays.

Thus, today spectrum scarcity and deployment delays are two of the major problems confronting wireless communications with regards to spectrum use [9]. Both these problems are

a result of the static nature of the current spectrum allocation strategy. Absence of flexibility in the spectrum allocation strategy makes it inefficient and the deployment process becomes complicated and time consuming.

The solution to these problems is based on the observation that, in a typical situation there are a number of instances of the allocated spectrum being in use only in certain geographical areas and again a number of instances of the allocated spectrum being in use only for brief periods of time. Studies have determined that even simple use of such *unused* spectrum will result in orders of magnitude improvement in the available capacity [9].

To address the above issues and to efficiently utilize the unused spectrum, DARPA's XG communications program is developing a new Spectrum access technology called *Dynamic Spectrum Access* (*DSA*).

The basic concept of DSA is that when a device wants to transmit, it senses the environment for the presence of the primary user in the spectrum that it wants to use. Using this sensed information, its current state information and the applicable regulatory policies, the device refrains from transmitting or decides to transmit in a manner that limits the level of interference. Thus basically, DSA delegates the spectrum management functions to each radio instead of having a centralized spectrum management as done in the legacy systems.

2.2 XG Radio Architecture

The XG Architecture [11] consists of Radio Hardware, Firmware and the Software components. At the highest level of abstraction it consists of the *Sensors*, the *RF module*, the *System Strategy Reasoner* (*SSR*) and the *Policy Reasoner*(*PR*) as shown in the Figure 2.1.

- Sensors: The sensors are used to sense the environment and discover the unused spectrum.
- Radio Frequency (RF): The RF module is used to send and receive transmissions.
- System Strategy Reasoner: The SSR is a radio hardware specific component and controls the radio's transmissions. It is responsible for generating optimized transmission requests based on the data received from sensors, the radio's current state and the radio's current strategies. The SSR then sends this transmission request to the PR to check if the request is legal with reference to the active policies.
- Policy Reasoner: The PR is a platform independent module. It works as a local policy decision unit for every XG Radio. It receives requests from the SSR, evaluates these requests based on the set of Active Policies. Once the transmission request evaluation is complete, the PR returns a reply which is either "Allow Transmission", "Deny Transmission" or "Transmission Request is Incomplete". In case the transmission request is denied or incomplete, the PR also returns opportunity constraints which if satisfied by the radio will allow transmission.

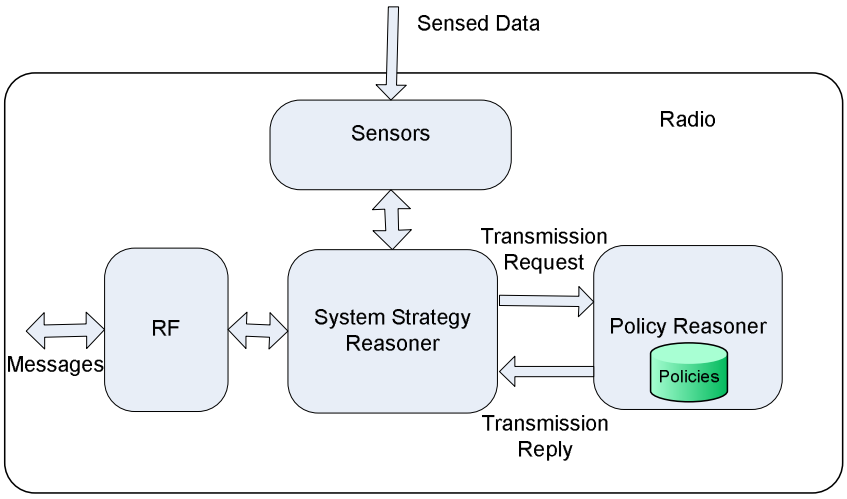

Figure 2.1: XG Radio Architecture

Three types of messages are described for interaction between the various modules. These messages are as below:

- RF - SSR: All the messages coming in and going out of the XG Radio pass through the RF Module. These messages include the control messages, the policy updates, etc. The RF Module passes on these messages to the SSR which can then act on these messages. All the control or data message generated in the SSR leave the radio through the RF module.

- SSR - Sensors: The sensors are responsible for sensing the environment and communicating any spectrum opportunities to the SSR. The SSR also may send control messages to the sensors.

- SSR - PR: There are two basic types of messages for communication between the SSR and PR. The first type of message is the Policy Update message. Using this message,

the SSR may ask the PR to update, add, remove or change the active policies. The PR can then reply with a Policy Update Acknowledge message. The second type of message from the SSR to the PR is the Transmission Request message. The PR evaluates this transmission request and returns a reply which may be either "Allow transmission", "Deny Transmission" or "Transmission Request is Incomplete". In case the Policy Reasoner denies transmission or the Transmission Request is incomplete, it will also send back opportunity constraints which if satisfied, transmission can be allowed.

2.3 Policy Language and Policies

The main requirements for the policy language as stated in [4] are accreditability, extensibility and expressiveness. Accreditability requires that it should be possible to accredit the policies independently of the Policy Reasoner and the radio. Extensibility requires that, it should be easy to add new domains to the policies while expressiveness demands that the policy language should be capable of satisfying the requirements of any new domain introduced. The XG Policy language was defined with an objective of achieving these three requirements.

The XG policy language is a declarative, semantic language which allows regulators to author rules to limit the interference in the Policy based dynamic spectrum access Networks. The XG policy language uses Semantic Web Rule Language First Order Logic (SWRL FOL) [33], which is an extension of the Semantic Web Rule Language (SWRL) [13], which in turn is

based on W3C Web Ontology Language (OWL) [12].

The XG Language provides a machine independent knowledge representation technique, such that the knowledge represented can be understood by both the machines and humans. It has the means to express facts and current state of the radio as well as the radio environment such that these facts and state information can then be used to check for policy conformance.

The policy language uses the concepts of permissions and prohibitions to express the actions that the device can undertake. Two types of policies are used viz. permissive policies and prohibitive policies. The permissive policy expresses the states in which the policy is applicable to the device and the constraints the device must satisfy in order to allow the transmission. Similarly the prohibitive policy defines the states and the conditions in which the device is prohibited from transmitting. A default de-confliction rule of the language states that the prohibitive policy always overrides a permissive policy. This means that in case a prohibitive policy and a permissive policy are satisfied at the same time, the transmission will be prohibited.

The parameters that constitute the spectrum access policies can be classified as given in [2].

1. Sensed Parameters: These parameters include the parameters that are sensed by the sensor module and reported to the Policy Reasoner. These parameters include the parameters like the Sensed Signal Levels, Node Location, Time, etc.

2. Parameters proposed by Radio: These parameters are the dependent on the radio strategy and are proposed by the System Strategy reasoner. These parameters include

the Transmit Power, Frequency, Modulation Type etc.

3. Environmental parameters: These are the parameters like Node Identity, Device Capability etc.

Policies can use a combination of any of these parameters. Some simple examples of the policies from [7] are as follows:

1. Frequency band - "Allow transmission between 5180 MHz and 5250 MHz."
2. Time - "Disallow transmission between 08:00 and 10:00 local time."
3. Location - "Allow transmission if radio is at most 30 miles away from the geographic coordinates (39 10' 30" N, 75 01' 42")."
4. Node Identity - "Allow transmission if radio belongs to the Red Cross."
5. Sensed Data Listen Before Talk - "Allow transmission if radio's peak sensed power is at most -80 dBm and it's EIRP for transmission is at most 10 mW."

As can be seen from the above examples some policies are based solely on the environmental parameters; some are based on the parameters proposed by the radio; some are based on the sensed parameters while some policies use a combination of all the above types of parameters. Any policy that uses sensed parameters is called as a Listen before Talk Policy [3]. Listen before Talk policies require the sensors to sense the environment and submit the data about the signals detected to the reasoner.

2.4 Spectrum Policy Representation

The spectrum access policies are XML based and hence they lend themselves to various forms of representations. In [27] authors have conducted a comparative analysis of the various methods for representing XML based policies.

Table 2.1: Policy representation methods

	Scalability	Tools	Policy Merging	Conflict Analysis
DNF	poor	NA	NA	NA
AOG	poor	NA	weak	complex
Alloy	poor	limited	NA	simple
RMM	NA	no	NA	NA
MTBDD	good	powerful	powerful	simple

In the Table 2.1 we combine their results with our own analysis of policy representation techniques. The likeliness between the policies and boolean functions makes it an attractive option to use the Disjunctive Normal Form (DNF) to represent the policies, however the authors in [27] claim that the DNF form of representation explodes in terms of memory and does not scale well beyond all but the simplest policies. Another natural choice to represent XML based policies is Alloy [30]. Alloy is a declarative specification language for expressing complex structural constraints and behavior in a software system and it provides a simple structural modeling tool based on First Order Logic (FOL). However Alloy faces some potential pitfalls. Authors in [27] claim that in their experiments with Alloy, the tool faced severe scalability problems while representing the policies. And-Or Graph(AOG) is another

policy representation method. In our experiments with AOGs, we found that handling conflict resolution and policy merging in AOGs was complicated and not scalable. Another option for policy representation is the Role-based access control Modal Model (RMM) [29]. However, this method does not provide any tool support. *Multi-terminal Binary Decision Diagrams* (*MTBDD*) have been proven to be simple and efficient for representing, processing and combining XML based policies [27] [28]. In [27], the authors observed that while in the worst case the number of nodes in an MTBDD is exponential, in practice, the number of nodes is often polynomial or even linear. Our experience with the spectrum access policies indicates that the size of MTBDD representation of the merged policies is polynomial in *most cases*.

2.5 Binary Decision Diagrams and Multi-terminal Binary Decision Diagrams

A Binary Decision Diagram (BDD) is a data structure used to represent Boolean functions. Every Boolean function can be represented as a rooted, directed, and acyclic graph which consists of two types of node viz. terminal nodes and non-terminal nodes or decision nodes. The two terminal nodes are called as 0-terminal and 1-terminal. Both the terminal nodes have a zero out-degree. Each non-terminal node or decision node, u, is labeled by a Boolean variable, $var(u)$, and has two child nodes called a low child, $low(u)$ and a high child, $high(u)$. An edge from a node to a low (high) child represents an assignment to the variable of 0(1).

In a figurative representation of a BDD, an edge to the lower child is represented using a dotted line while an edge to the higher child is represented using a solid line.

A BDD is called as an *ordered BDD* or an *OBDD*, if the variables appear in the same order on all the paths from the root to the terminal [10]. Also, a BDD is said to be *reduced* if the following two rules have been applied to its graph:

- No variable node u has identical low- and high- successors, i.e.,

  $high(u) \neq low(u)$.

- Uniqueness: No two distinct nodes u and v have the same variable name and the same low- and high-successors, i.e.,

  $low(u) = low(v)$, $high(u) = high(v)$, and $var(u) = var(v)$ implies that $u = v$.

The reduction rules can be shown as in Figure 2.2.

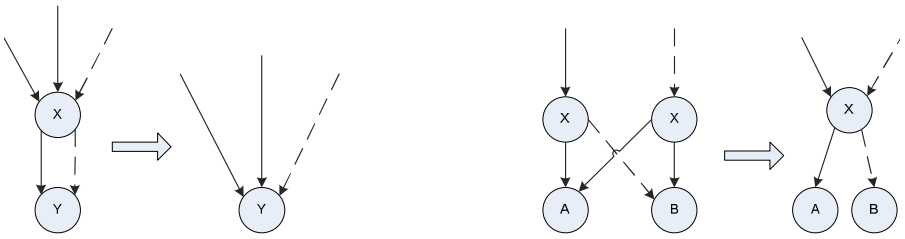

Figure 2.2: Reduction Rules for Binary Decision Diagrams

In popular usage the term BDD almost always refers to *Reduced Ordered Binary Decision Diagram* (*ROBDD*). The advantage of an ROBDD is that it is unique for a particular functionality. The size of the BDD is determined both by the function being represented and the chosen variable ordering.

Multi-terminal Binary Decision Diagrams (*MTBDD*) are more general form of BDDs [14]. They extend BDD such that they can represent functions of an arbitrary range, while their domain is still a multidimensional Boolean space. i.e., an MTBDD provides a compact representation of functions of the form $P : B^n \rightarrow E$, which maps bit vectors over a set of variables,B^n, to a finite set of results E. So the structure of an MTBDD is the same as a BDD with the difference that MTBDDs have more than two terminal nodes.

2.6 Related Work

Recently, significant amount of work has been done on Dynamic Spectrum Access and spectrum access policies in academia as well as the industry. The DARPA XG program aims to develop the concepts, framework and enabling technologies for DSA. Under the auspices of DARPA's XG program, BBN Technologies developed an XG policy language framework which was based on OWL [9] [16]. The Shared Spectrum Company has developed policy-based spectrum access control systems [5] [6]. Also, SRI International has been involved in developing their own XG policy Language framework and policy based cognitive radio systems [4] [7] [11] [26].

In [7], SRI International introduce a policy language called Cognitive Radio Language(CoRaL), which is a cognitive radio specific, logic based, specification language and is different from BBN's policy language which is based on OWL. SRI International have also developed a policy engine which *reasons* using the CoRaL policy language. This policy engine uses a

Prolog reasoning engine. However, a major shortcoming of this reasoner is that it returns only "Yes/No" replies and does not give back any opportunity constraints.

In [4], SRI International has developed a Policy reasoner which uses a *Maude reasoning engine*. This reasoner uses policies defined in SWRL FOL/SWRL/OWL policy languages. Policies and facts written in these languages are converted to a specification language called Maude for reasoning. The Maude reasoner returns "Yes/No" answer to the transmission request. It also returns back opportunity constraints to the SSR in case of denied transmission or an incomplete transmission request. This reasoner uses a combination of forward chaining, backward chaining, partial evaluation based on conditional rewriting, and constraint propagation and simplification. However from our experience of testing the Maude reasoner, we have observed that the reasoner returns back all possible opportunity constraints. In one of the experiments with 11 active spectrum access policies, the reasoner returned 75 opportunity constraints and it took 58 seconds to return this reply.

In [5] [6], Shared Spectrum Company has been developing policy-based spectrum access control systems. In field demonstrations, they show that declarative spectrum management is operationally practicable as the field tested radios were making correct runtime decisions and that the approach could be used to enforce requirements of multiple stake holders. However, the demonstrations also illustrated that the policy reasoner can incur large memory overhead and processing delay unless its design and implementation are not optimized to reduce such overhead.

In [17] [18], the XG Policy language [16] is extended based on a *Community Based Policy*

Management (*CBPM*) mechanism, which explores an community hierarchy structure for defining and enforcing policies. It provides a way to organize the spectrum policies in an authority hierarchical architecture and resolve policy conflicts from multiple authorities.

The issue of Policy Conflict analysis and resolution has been well researched in conventional policy-based network management systems, e.g. policy conflict analysis in Distributed management system [19] [20], for QoS management policies [21] [22] [23], for network security policies [24] [25] and for network access policies [27] [28]. In [27], the researchers have developed a software tool called Margrave for the analysis of role-based access-control policies. Margrave represents policies using the MTBDDs.

Chapter 3

BRESAP

3.1 BRESAP Architecture

BRESAP processes spectrum access policies using "Multi Terminal Binary Decision Diagrams". Figure 3.1 illustrates the BRESAP architecture. BRESAP receives the Spectrum access policies defined in SWRL FOL/SWRL/OWL languages. The *Policy Parser* module parses these policies and represents them in the form of boolean expressions using *Boolean Encoding*. The *Policy Preprocessor* module then processes these boolean expression in context of their semantics to generate *Auxiliary rules* which are additional constraints represented as boolean expressions. These boolean expressions are then converted by the *Policy Converter* module into BDDs. The *Policy Merger* module finally merges all the individual BDDs representing the policies and the auxiliary rules to create a *meta-policy MTBDD*. It

further reduces the meta-policy MTBDD to a *meta-policy BDD*.

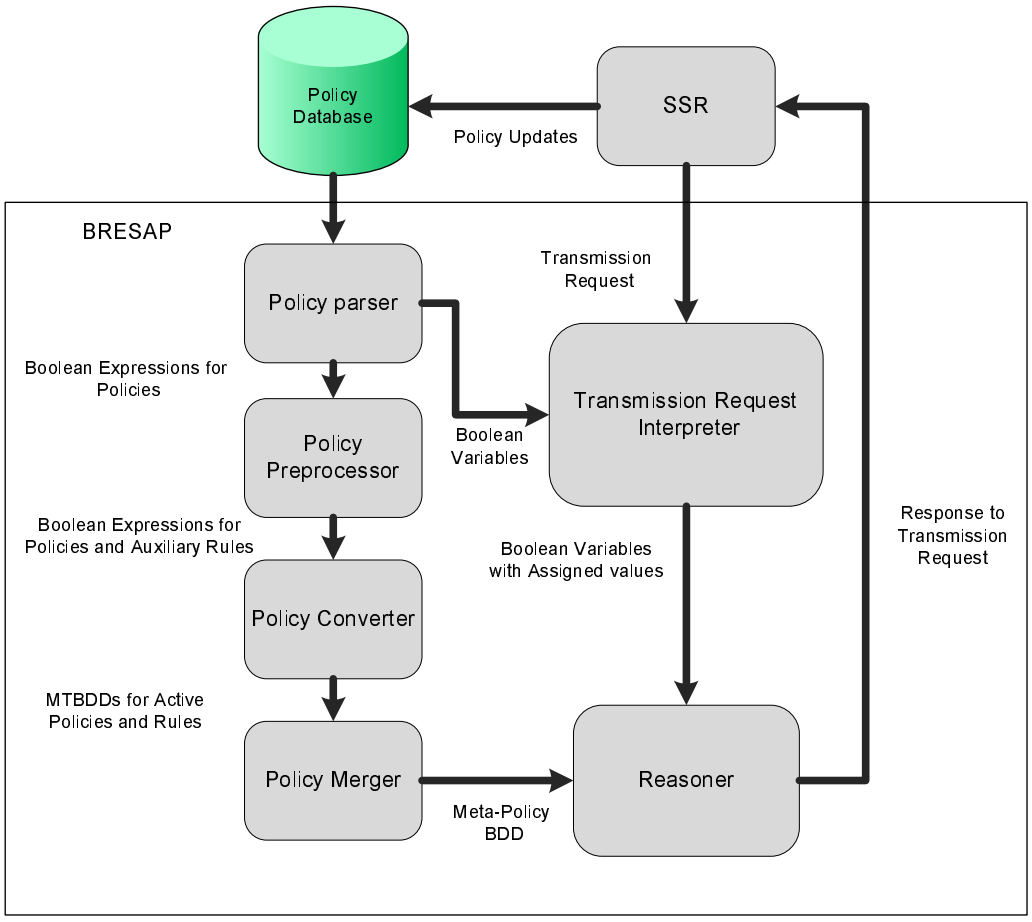

Figure 3.1: BRESAP Architecture

When the SSR is ready with a transmission request it sends it over to the Policy Reasoner. The *Transmission Request Interpreter* module parses the transmission request and processes it. The *Reasoner* module uses the data from the transmission request interpreter and the meta-policy BDD to evaluate the Transmission Request and send back the reply to the SSR.

In what follows, each BRESAP component and its functionality are described in detail.

3.2 Policy Parser

The policy parser module converts the SWRL FOL policies [1] into Boolean expressions using *Boolean Encoding*.

The SWRL FOL policies are based on *attributes* which are the various parameters of Dynamic Spectrum Access domain; for example, Center Frequency, Transmit Power, Sensed Power, Time are some of the attributes in DSA. These attributes have values bound to them, called as, *attribute Values*. A fragment of an SWRL FOL policy designating the Center Frequency as 231000000 Hz is as shown below:

$< swrlx : datavaluedPropertyAtom\ swrlx : property =$“xg;centerFrequency”$>$

$< ruleml : var > tr < /ruleml : var >$

$< owlx : DataValue\ owlx : datatype =$“xsd;integer”$>$ 231000000 $< /owlx : DataValue >$

$< /swrlx : datavaluedPropertyAtom >$

This fragment represents an *attribute-value pair*. During the process of Boolean encoding, the parser represents each unique attribute-value pair in a policy using a distinct variable. The variables are then combined using the logical operations $\vee$ and $\wedge$ depending on the policy structure to create a boolean expression that represents the policy. Thus, the parser transforms policy P into the function $P : B^n \rightarrow E$, which is a mapping from a set of Boolean variables, B^n onto the finite set of effects, $E = \{Y, N, NA\}$. The functionality of the parser

can be better explained with an example given below. Note that all the policies hence forth are given in text format for simplicity purpose.

Consider two simple spectrum access policies:

- Policy 1: Allow transmission if the Center Frequency is $245MHz$ and Transmit Power is less than $25dbm$.
- Policy 2: Deny transmission if the Sensed Power is greater than $-100dbm$.

The policy parser uses boolean encoding to map each policy constraint to an Atomic Boolean expression AE_i, represented by a variable x_i as below: $x_1 : (Center\ Frequency = 245\ MHz)$
$x_2 : (Transmit\ power \leq 25\ dbm)$
$x_3 : (Sensed\ power \geq -100\ dbm)$

In the next step, the logical operators $\vee$ and $\wedge$ are used to create boolean expressions as shown below:
Policy 1: *Allow Transmission* if $(x_1 \wedge x_2)$
Policy 2: *Deny Transmission* if (x_3)

The BRESAP reasoning algorithms depend on a very important property of Boolean encoding. The property is as stated below.

Property 1 *In a Boolean representation of a spectrum access policy, all literals are positive,*

i.e., their complements do not appear in the Boolean expression.

A straightforward Boolean representation of an arbitrary cognitive radio policy may not satisfy Property 1. However, BRESAP was designed to *reason* using policies represented as Boolean expressions that satisfy the property. Hence any representation of a policy first needs to be translated into a Boolean representation satisfying Property 1 before the PR can begin processing the policies.

These Boolean expressions are then passed on to the Policy Preprocessor module which is explained in Section 3.5.

3.3 Policy Converter

The policy converter module uses the *Build* algorithm [10], to convert boolean expressions into BDDs. The Build algorithms in turn uses the *Make* algorithm to create new nodes during BDD generation. The Make algorithm ensures that the BDD created, is in "Reduced" form as explained in Section 2.5. The run time of the make algorithm is $O(1)$. This is achieved by using hash maps for the purpose of storing the newly created nodes. The build algorithm however is recursive in nature and its run time is exponential. If a Boolean expression contains n variables, then the runtime of build algorithm is of the order $O(2^n)$.

BDDs are rooted, acyclic and directed graphs with two terminal nodes. In case of BRESAP, these terminal nodes may be of three different types viz. *No* (N), *Not Applicable* (NA)

and $Yes(Y)$. In BDDs, the non-terminal nodes represent variables for the attribute-value pairs as assigned by the policy parser and the terminal nodes represent the values in the policies set of effects. Each path in the BDD from the root node to the terminal node represents an assignment of values to the variables along the path. Any given transmission request corresponds to a variable assignment, which in turn corresponds to a path. A complete transmission request would correspond to a complete path and an underspecified transmission request would correspond to an incomplete path. For a complete transmission request, the terminal that is reached on the corresponding path gives the effect of the policy for that request. In the Figure 3.2, dashed lines denote $0-edges$ (assignment of 0 to the Boolean variable) and solid lines denote $1-edges$ (assignment of 1 to the Boolean variable).

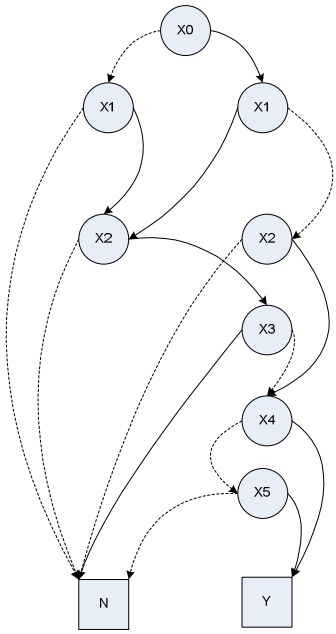

Figure 3.2: BDD

Note that different orderings of the variables may result in different BDD representations of

different sizes. During the building of the BDDs, BRESAP uses a fixed variable ordering which is $x_0 < x_1 < ... < x_{n-1}$, where n is the number of variables in the BDD.

The BDDs created by the policy converter module for the example policies in Section 3.2 are as below:

Policy 1: *Allow Transmission* if $(x_1 \wedge x_2)$

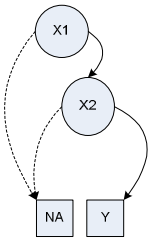

Figure 3.3: Example Policy 1

Figure 3.3 shows the BDD for Policy 1. This BDD has two terminals viz. *Not Applicable* (NA) and *Yes*(Y). The policy $P1$ is a "permissive" spectrum access policy. When all the constraints in this policy are satisfied and Y terminal is reached, the transmission is allowed. However, if any of the constraints in this policy is not satisfied and NA terminal is reached, this policy is considered as "Not Applicable" to that scenario and is ignored. Thus, the Y terminal represents the state of "Allow Transmission" while the NA terminal represents the state of "Ignore Policy".

Policy 2: *Deny Transmission* if (x_3)

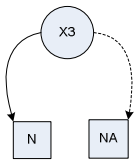

Figure 3.4: Example Policy 2

Figure 3.4 shows the BDD for Policy 2. This BDD has two terminals viz. *No* (N) and *Not Applicable* (NA). The policy $P2$ is a "prohibitive" spectrum access policy. If all the constraints in this policy are satisfied and N terminal is reached, the transmission is denied. However, if any of the constraints in this policy is not satisfied, the policy is considered as "Not Applicable" to that scenario and is ignored. The terminal N, thus, represents the state of "Deny Transmission" while the terminal NA represents state of "Ignore Policy".

Thus we see that, a BDD which represents a permissive policy will have the two terminal nodes NA and Y, while a BDD which represents a prohibitive policy will have the two terminal nodes N and NA.

3.4 Policy Merger

Policy merger module combines all the individual BDDs generated by the policy converter, to build a single *Meta-policy BDD*. BRESAP uses the "Apply" operation defined in [10] to combine the individual BDDs into a single Meta-policy MTBDD which is then reduced to a meta-policy BDD.

The Apply algorithm combines two BDDs by using a pre-defined binary arithmetic operation.

"Apply" traverses each of the BDDs simultaneously starting from the root node down towards the terminal nodes. A variable ordering needs to be specified in order to execute the Apply operation. Assuming x_i and x_j are nodes to be combined in the two BDDs and the variable ordering used is $x_i < x_j$ when $i < j$, there can be two cases for the $non-terminal$ nodes of the two BDDs, viz. $i = j$ and $i < j$. The following two equations show the rules that *Apply* uses to combine the $non-terminal$ nodes.

The case $i = j$ in effect means that same variables from the two BDDs are being combined.In this case the Apply operation used is :

$Apply(Node(x_i, low_1, high_1), Node(x_i, low_2, high_2)) =$

$Node(x_i, Apply(low_1, low_2), Apply(high_1, high_2)$

Similarly, the case $i < j$ means that the variable x_i has a lower order as compared to x_j. The Apply operation used for combining decision nodes in such a case is:

$Apply(Node(x_i, low_1, high_1), Node(x_j, low_2, high_2)) =$

$Node(x_i, Apply(low_1, Node(x_j, low_2, high_2)), Apply(high_1\ , Node(x_j, low_2, high_2))$

When the terminal nodes of the BDDs are reached, the pre-defined binary arithmetic operation which we call as the *Terminal Node Combination Operation* is applied to obtain a resulting terminal node. In a reasoner, conflict occurs when the response for two different policies is different for the same request. The specified "Terminal Node Combination Operation" also determines the conflict resolution policy of the reasoner. BRESAP uses the *Deny*

Over-ride Rule to resolve policy conflicts. In a deny over-ride rule, the reasoner denies the transmission request if any of the active prohibitive policies denies the transmission.

The *Terminal Node Combination Operation* that BRESAP uses is as below :

$NA\ and\ NA \rightarrow NA \ldots (1)$

$NA\ and\ Y \rightarrow Y \ldots (2)$

$NA\ and\ N \rightarrow N \ldots (3)$

$N\ and\ Y \rightarrow N \ldots (4)$

Equation (1), states that an "ignore" terminal from one BDD when combined with the "ignore" terminal from second BDD results in "ignore" terminal in the final MTBDD. Equation (2), states that an "ignore" terminal from one BDD when combined with the "Yes" terminal from second BDD should result in "Yes" terminal in the final MTBDD. Equation (3), states that an "ignore" terminal from one BDD when combined with the "No" terminal from second BDD should result in "No" terminal in the final MTBDD. Equation (4), states that an "No" terminal from one BDD when combined with the "Yes" terminal from second BDD should result in an "No" terminal in the final MTBDD.

Figure 3.5 shows the meta-policy BDD generated by the policy merger module from the individual BDDs in Section 3.3

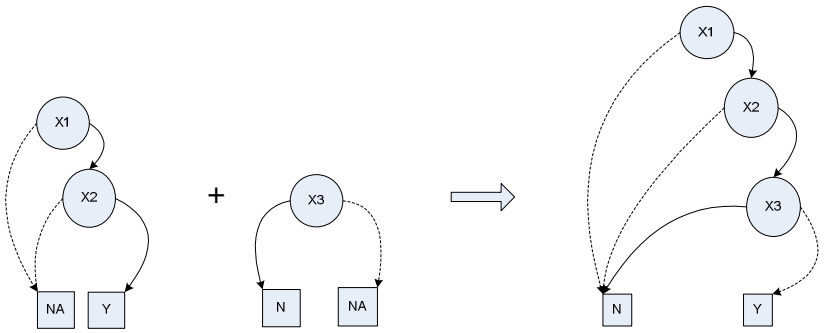

Figure 3.5: Policy Merging

As can be seen from figure, the transmission is allowed only in the case where all the constraints in the permissive policy are satisfied and one or more constraints in the prohibitive policy are not satisfied.

The NA terminal plays a very important role during the policy merger as seen from the Terminal Node Combination Approach. However, once all the active policies have been combined to create a meta-policy MTBDD, there is no difference between the N and the NA terminal nodes. This is because all the transmission requests whose paths terminate either at the N or the NA terminal are denied by the Policy Reasoner. As a result of this, once the meta-policy MTBDD is generated we combine the N and NA terminals of the Meta-MTBDD, to convert the MTBDD into BDD. This node combination is advantageous as it results in decreasing the number of nodes in the final meta-policy and lets us use a number of efficient graph - theoretic algorithms that work only on BDDs. An example conversion of an MTBDD into a BDD by combining the N and NA nodes is shown, wherein the MTBDD in Figure 3.6 is converted to a BDD in Figure 3.7.

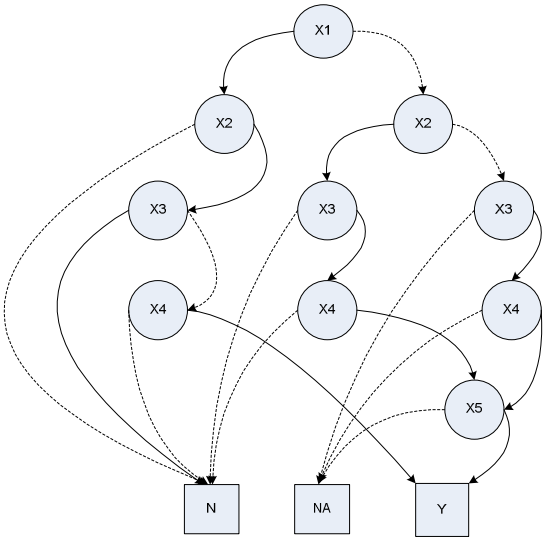

Figure 3.6: Node Elimination example

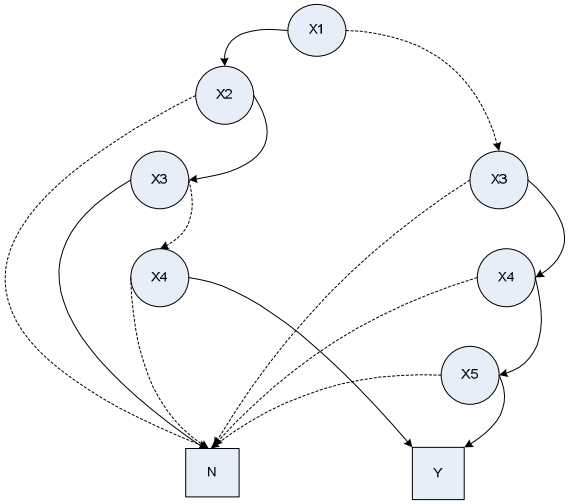

Figure 3.7: Node Elimination example

The BDD in Figure 3.7 can then be used for reasoning by the *Policy Reasoner*.

3.5 Policy Preprocessor

When merging two BDDs into one, we need to analyze the Boolean expressions represented by the nodes of each BDD prior to merging. Without such analysis, the resulting BDD after merging may contain illogical paths. Such paths in the meta-policy BDD can cause the Reasoner module to compute illogical opportunity constraints. For example, consider the meta-policy MTBDD shown in the below figure:

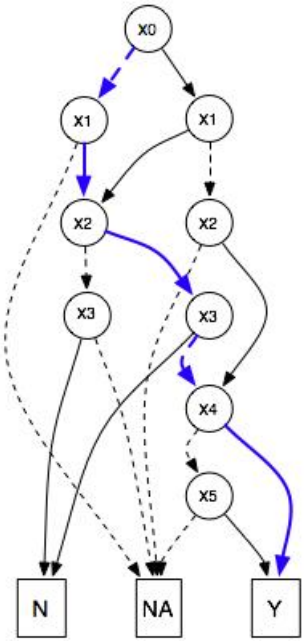

Figure 3.8: MTBDD

Let us assume that in this Meta-MTBDD, the Boolean Encoding used is as follows:

$x_1 : (Transmission\ Power \leq -10\ dbm)$

$x_2 : (Center\ Frequency = 235\ MHz)$

$x_3 : (Sensed\ Power \leq -110\ dbm)$

$x_4 : (Center\ Frequency = 255\ MHz)$

$x_5 : (Operation\ Mode = Day\ to\ Day)$

The blue colored path from the root node to the terminal Y corresponds to an assignment of values to the parameters as shown below:

$x_0 \rightarrow 0$

$x_1 \rightarrow 1$

$x_2 \rightarrow 1$

$x_3 \rightarrow 0$

$x_4 \rightarrow 1$

However, x_2 and x_4 can never be true at the same time because, at any given time the Center Frequency can either be 235 MHz or 255 MHz but not both. Thus, the path marked in blue color is illegal. Existence of such illogical paths can cause the reasoner to return illogical opportunity constraints and hence such illogical paths must be removed from the meta-policy MTBDD. This is the function of the *Policy Preprocessor*. The preprocessor *semantically* analyzes the Boolean expressions generated by the Policy parser module and generates additional constraints (boolean expressions) which we call as *Auxiliary Rules*, such that, when these auxiliary rules are combined with the Meta-policy MTBDD, they eliminate all the illogical paths from the Meta-MTBDD.

These auxiliary rules represent the *Environmental Constraints* for Dynamic Spectrum Access and the Preprocessor module is thus responsible for enforcing the environmental constraints

by eliminating the illegal paths which violate them. Currently we do not consider cross domain environmental constraints, so BRESAP uses one Auxiliary rule per domain to constrain all the parameters in that particular domain according to the *laws of physics*.

BRESAP currently handles six parameters viz.

1. Center Frequency
2. Time
3. Sensed power
4. Transmit Power
5. Distance
6. Operation Mode

The derivations for auxiliary rules for these parameters are as below.

3.5.1 Auxiliary Rule for Center Frequency

Let us assume that a Meta-policy MTBDD has two variables x_1 and x_2 representing center frequency constraints. Let, the boolean encoding used be :

$x_1 : (Center\ Frequency = 235\ MHz)$

$x_2 : (Center\ Frequency = 255\ MHz)$

In this case, the meta-policy MTBDD should not have a path from the root terminal to the "Y" terminal node with a node assignment of "true" for both x_1 and x_2. Any such existing path needs to be directed to the "N" terminal. The preprocessor creates an Auxiliary rule which when combined with the meta-policy MTBDD directs all such paths violating the center frequency constraint to the "N" terminal. The Auxiliary rule created for the given example is:

$Disallow\ if : (x_1 \wedge x_1)$

Let f_1, f_2,...,f_k be variables in the meta-policy MTBDD representing distinct center frequency constraints. Then the auxiliary rule for the Center Frequency can be generalized as:

$Disallow\ if : ((f_1 \wedge f_2) \vee (f_1 \wedge f_3) \vee ... (f_1 \wedge f_k) \vee (f_2 \wedge f_3) \vee ... (f_2 \wedge f_k) \vee (f_3 \wedge f_4) \vee ... \vee ... (f_{k-1} \wedge f_k))$

3.5.2 Auxiliary Rule for Operation Mode

The general form of Auxiliary rule for operation mode is same as the auxiliary rule for center frequency as the environmental constraints for these two are similar. Let f_1, f_2,...,f_k be variables in the meta-policy MTBDD representing distinct operation modes constraints. Then the auxiliary rule for the operation mode can be generalized as:

$Disallow\ if : ((f_1 \wedge f_2) \vee (f_1 \wedge f_3) \vee ... (f_1 \wedge f_k) \vee (f_2 \wedge f_3) \vee ... (f_2 \wedge f_k) \vee (f_3 \wedge f_4) \vee ... \vee ... (f_{k-1} \wedge f_k))$

3.5.3 Auxiliary Rule for Transmit Power

Let us assume that a Meta-policy MTBDD has four variables x_1, x_2, x_3, x_4 representing Transmit Power constraints. Let the boolean encoding used be:

$x_1 : (Transmit\ Power \leq 20\ dbm)$

$x_2 : (Transmit\ Power \leq 40\ dbm)$

$x_3 : (Transmit\ Power \leq 60\ dbm)$

$x_4 : (Transmit\ Power \leq 80\ dbm)$

The environmental constraints on the Transmit Power parameter in the given case can be listed as below:

1. When x_1 is true, none of x_2, x_3 and x_4 can be false.

2. When x_2 is true, none of x_3, x_4 can be false, however, x_1 can be false.

3. When x_3 is true, x_4 cannot be false, however, x_1 and x_2 can be false.

4. When x_4 is true x_1, x_2 and x_3 can be false.

The preprocessor creates an auxiliary rule which when combined with the meta-policy MTBDD directs all such paths violating the Transmit Power constraint to the "N" terminal. The Auxiliary rule created for the given example is:

$Disallow\ if : ((x_1 \wedge \sim x_2\ ) \vee (x_1 \wedge \sim x_3\ ) \vee (x_1 \wedge \sim x_4\ ) \vee (x_2 \wedge \sim x_3\ ) \vee (x_2 \wedge \sim x_4\ ) \vee (x_3 \wedge \sim x_4\ ))$

Let f_1, f_2,...,f_k be variables in the meta-policy MTBDD representing distinct Transmit Power

constraints in ascending order. Then the auxiliary rule for the Transmit Power can be generalized as:

$Disallow\ if : ((f_1 \wedge \sim f_2) \vee (f_1 \wedge \sim f_3) \vee ... (f_1 \wedge \sim f_k) \vee (f_2 \wedge \sim f_3) \vee (f_2 \wedge \sim f_k) \vee (f_3 \wedge \sim f_4) \vee ..(f_3 \wedge \sim f_k) \vee ... (f_{k-1} \wedge \sim f_k))$

3.5.4 Auxiliary Rule for Sensed Power

Auxiliary rule for Sensed power is same as that for Transmit power as the environmental constraints for these two parameters are similar. Let f_1, f_2,...,f_k be variables in the meta-policy MTBDD representing distinct Sensed Power constraints in ascending order. Then the auxiliary rule for the Sensed Power can be generalized as:

$Disallow\ if : ((f_1 \wedge \sim f_2) \vee (f_1 \wedge \sim f_3) \vee ... (f_1 \wedge \sim f_k) \vee (f_2 \wedge \sim f_3) \vee (f_2 \wedge \sim f_k) \vee (f_3 \wedge \sim f_4) \vee ..(f_3 \wedge \sim f_k) \vee ... (f_{k-1} \wedge \sim f_k))$

3.5.5 Auxiliary Rule for Time

The BRESAP parser specifies time using two sided intervals. For example:

$Interval\ 1 : x_1 => (t \geq 1\ pm)\ and\ x_2 => (t \leq 3\ pm)$

$Interval\ 2 : x_3 => (t \geq 2\ pm)\ and\ x_4 => (t \leq 5\ pm)$

$Interval\ 3 : x_5 => (t \geq 8\ pm)\ and\ x_6 => (t \leq 12\ pm)$

To create an auxiliary rule for the time domain, the preprocessor module first finds all the

independent time intervals. The Meta-policy MTBDD should not have a path from the root terminal to the "Y" terminal node with a node assignment of "true" to two independent time intervals. Any such existing path needs to be directed to the "N" terminal. The preprocessor creates an Auxiliary rule which has this effect on the Meta-MTBDD. In the above example we see that Interval 1 and 2 are intersecting while Intervals 1 and 3 and Intervals 2 and 3 are Independent. So the preprocessor generates below constraints to be added to the meta-policy MTBDD.

1. *Interval* 1 and *Interval* 3 cannot be true at the same time.

2. *Interval* 2 and *Interval* 3 cannot be true at the same time.

So the auxiliary rule that the preprocessor generates is as follows :

$Disallow\ if : ((x_1 \wedge x_2) \wedge (x_3 \wedge x_4)) \vee ((x_3 \wedge x_4) \wedge (x_5 \wedge x_6\ ))$

3.5.6 Auxiliary Rule for Location

BRESAP assumes that the spectrum access policies define location parameter as a circular region around a particular point. For example, a location parameter x_i might be defined as the circular region around the point "73.45; 75.77; 0.0" with a radius of 20 miles, where 73.45 is the latitude, 75.77 is the longitude while 0.0 is the altitude of the centre. Assume that a Meta-policy MTBDD has 3 location parameters x_1, x_2, x_3. Also assume that, regions

represented by the boolean variables x_1 and x_2 intersect with each other, however regions represented by x_1 and x_2 do not intersect with region x_3.

To create an auxiliary rule for the location domain, the preprocessor module first finds all variables representing non-intersecting regions. The Meta-policy MTBDD should not have a path from the root terminal to the "Y" terminal node with a node assignment of "true" to two non-intersecting regions. Any such existing path needs to be directed to the "N" terminal. The preprocessor creates an Auxiliary rule which when combined with the Meta-MTBDD has this effect. In the above example the preprocessor generates below constraints to be added to the meta-policy MTBDD.

1. If x_1 is true then, x_3 cannot be true.

2. If x_2 is true then, x_3 cannot be true

3. If x_3 is true then, x_1 and x_2 cannot be true.

So the auxiliary rule that the preprocessor generates is as follows :

$Disallow\ if : ((x_1 \wedge x_3) \vee (x_2 \wedge x_3))$

Let f_1, f_2,...,f_k be variables representing k non-intersecting regions. Then the auxiliary rule for the location can be generalized as:

$Disallow\ if : ((f_1 \wedge f_2) \vee (f_1 \wedge f_3) \vee ... (f_1 \wedge f_k) \vee (f_2 \wedge f_3) \vee ... (f_2 \wedge f_k) \vee (f_3 \wedge f_4) \vee ... \vee ... (f_{k-1} \wedge f_k))$

3.5.7 Example

The working of the preprocessor can be illustrated with an example. Consider two policies as shown below:

$Policy\ 1 : Transmit\ if$: Center Frequency = 235 MHz and Transmission Power ≤ -20 dbm

$Policy\ 2 : Transmit\ if$: Center Frequency = 245 MHz and Transmission Power ≤ -10 dbm and Sensed Power ≤ -110 dbm.

Here the policy parser uses boolean encoding as given below:

$x_1 : (Center\ Frequency = 235\ MHz)$

$x_2 : (Transmission\ Power \leq -20\ dbm)$

$x_3 : (Center\ Frequency = 245\ MHz)$

$x_4 : (Transmission\ Power \leq -10\ dbm)$

$x_5 : (SensedPower \leq -110\ dbm)$

The two policy Boolean expressions generated by the policy parser are as below :

$Policy\ 1 : Allow\ if : (x_1 \wedge x_2)$

$Policy\ 2 : Allow\ if : (x_3 \wedge x_4 \wedge x_5)$

The BDDs for the two policies are as shown in Figure 3.9:

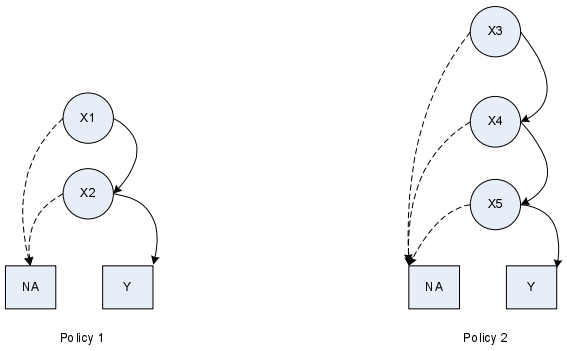

Figure 3.9: BDDs for the Policies

In this case the Policy preprocessor generates two Auxiliary rules one for the Center Frequency and one for Transmission Power. The two Auxiliary rules are as below:

Auxiliary rule for Center Frequency is:

$Disallow\ if : (x_1 \wedge x_3)$

Auxiliary rule for Transmission Power is:

$Disallow\ if : (x_2 \wedge \sim x_4)$

The BDDs for the two Auxiliary rules are as shown in Figure 3.10:

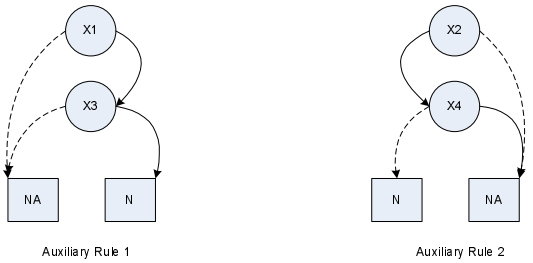

Figure 3.10: BDDS for the Auxiliary Rules

The meta-MTBDD for the two policies is as shown in Figure 3.11:

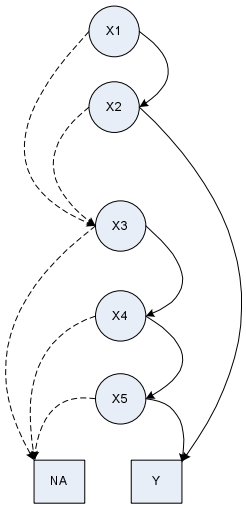

Figure 3.11: MTBDD after combination of Policy 1 and Policy 2

The meta-MTBDD when the two auxiliary rules are combined with the meta-MTBDD of the two policies is as shown in Figure 3.12.

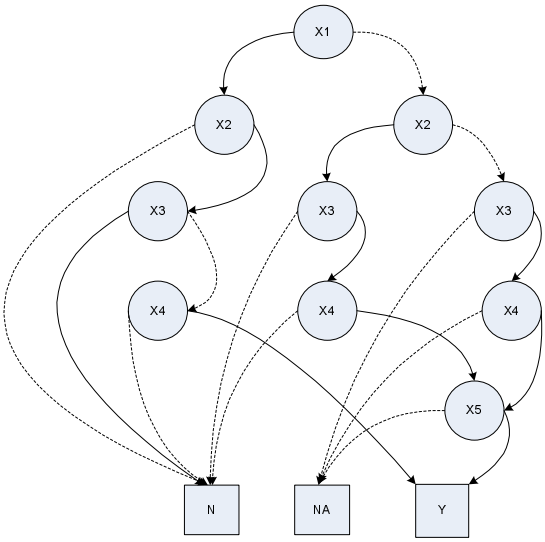

Figure 3.12: Final meta-policy MTBDD

As we can see from Figure 3.12, there is no illegal path to the "Y" terminal in the Meta-

MTBDD as all such paths have been directed to the "N" terminal node.

3.6 Transmission Request Interpreter

When the SSR is ready with a transmission request, it sends over the transmission request to BRESAP. The Transmission Request Interpreter module parses this request and processes it. The PR assumes that the transmission request submitted by the SSR has a structure as defined below:

Definition 1 *Let $a_1, a_2, ..., a_k$ represent a sequence of attribute names and $v_1, v_2, ..., v_k$ represent a sequence of values, where v_i is an value of the attribute a_i. The transmission request is defined as $r = \{(a_1, v_1), (a_2, v_2), \cdots, (a_k, v_k)\}$.*

An example of the request from the SSR is as follows:

$(Center\ Frequency : 232000000\ Hz)$, $(Location : 33.26; 77.56; 0.0)$, $(Transmission\ Power : -10\ dbm)$, $(Peak\ Sensed\ Power : -100\ dbm)$, $(Time : 2008 - 5 - 5\ 10 : 0 : 0.579)$

After the processing of Policy Merger module is complete, it generates a Meta-policy BDD as the output. In order to process the transmission request and compute a response, the policy reasoner needs that the boolean variables in the meta-policy BDD be assigned values of either 1 or 0. The Transmission Request Interpreter parses the request and analyzes it semantically. Using this data, it then assigns values of 1 or 0 to the Boolean variables in the meta-policy BDD depending on whether they are satisfied or unsatisfied.

For example, consider the meta-policy BDD from example in Section 3.4. The meta-MTBDD is as shown in the Figure 3.13:

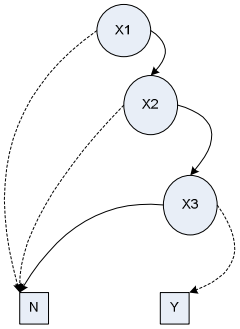

Figure 3.13: Transmission Request Interpreter Example

The Boolean encoding that BRESAP parser used here was:

$x_1 : (Center\ Frequency = 245\ MHz)$

$x_2 : (Transmit\ Power \leq -10\ dbm)$

$x_3 : (Sensed\ Power \geq -100\ dbm)$

Now suppose that the SSR sends a transmission request to the PR which is as follows:

$(Center\ Frequency : 245\ MHz)$,$(\ Transmit\ Power : -10\ dbm)$,$(Sensed\ Power : -110\ dbm)$

The transmission request interpreter evaluates each variable in the meta-policy MTBDD in context of this transmission request. It determines that the constraints represented by the boolean variables x_1 and x_2 are satisfied by the transmission request while the constraint represented by the boolean variable x_3 is not satisfied by the transmission request. Using this data the transmission request Interpreter assigns a value of 1 to x_1 and x_2 and it assigns

a value of 0 to x_3 as shown in the Figure 3.14.

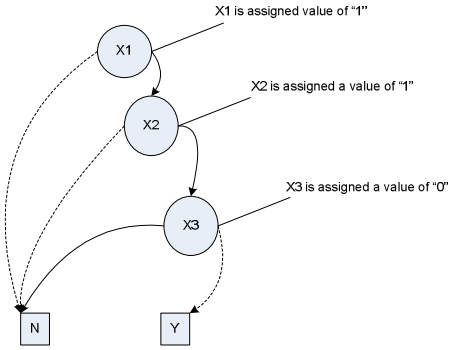

Figure 3.14: Transmission Request Interpreter Example

Note that it is not necessary that each of the Boolean variables will be assigned a value of 1 or 0. It is possible that the data provided by the transmission request is insufficient to assign values to all the Boolean variables. This would be a case of an incomplete transmission request.

3.7 Policy Reasoner

The Policy Reasoner module uses the meta-policy BDD and the Boolean interpretation of the transmission request to formulate a reply and send it back to the System Strategy Reasoner. The reasoner sends one of the following replies to the SSR:

- Transmission Allowed: This response is given for a valid transmission request.
- Transmission Denied: This response is given for an invalid transmission request. However, the reasoner also returns a set of opportunity constraints to the SSR, such that, if

the SSR applies these changes to the transmission request, the transmission is allowed.

- Transmission Request Incomplete: This response is given to an under-specified transmission request. However, the reasoner also returns a set of opportunity constraints to the SSR, such that, if the SSR applies these changes to the transmission request, the transmission is allowed.

The Policy Reasoner first uses the *FindPath algorithm* to traverse the meta-policy BDD to check if the transmission request is allowed or denied. If the "Y" terminal is reached during this traversal, the transmission is allowed. However, if the algorithm reaches the "N" or "NA" terminals, transmission is disallowed and one of the reasoning algorithms is invoked to find the node value assignments that will lead to the Y terminal. We have devised three different algorithms that can be used by the Reasoner module for computing opportunity constraints. These algorithms are:

- The *GreedyPath algorithm* that computes suboptimal opportunity constraints for an invalid transmission request.
- The *MinCostPath algorithm* that computes a single set of optimal opportunity constraints for an invalid transmission request.
- The *AllMinCostPath algorithm* that computes all the available sets of optimal opportunity constraints for an invalid or incomplete transmission request.

3.7.1 FindPath Algorithm

Once the Transmission Request Interpreter is done with assigning the values to the Boolean Variables in the meta-policy BDD, the first algorithm used is the FindPath Algorithm.This algorithm utilizes the observation that in BDDs, if a given node is not the N terminal node, it has at least one path leading to the Y terminal node. This algorithm follows the path from the root to one of the terminal nodes depending on the values of the Boolean variables it encounters on the way. If the value of the Boolean variable is 1, then the algorithm selects the high child of that node as its next destination and if the value of the Boolean variable is 0, then the algorithm selects the low child of that node as its next destination. The terminal node that is reached may be Y or N. If the terminal node reached is Y the algorithm sends back a reply to the SSR, saying that the transmission is allowed. However, if the N terminal is reached then the algorithm invokes one of the reasoning algorithms to return opportunity constraints to the reasoner. Also if one of the Boolean variables does not have an assigned value, then the algorithm terminates at that node and invokes one of the reasoning algorithms. This is the case of "Incomplete Transmission Request".

Assume that $\langle x_i \rangle$ is a value assigned to the Boolean variable x_i by the Transmission Request Interpreter during the process of evaluating a transmission request. The notations high(t) and low(t) represent the high-child and the low-child of node t, respectively. A pseudo code of the FindPath algorithm is given below.

Algorithm 1 FindPath

Require: BDD, Start Node, $\langle x_i \rangle$
Ensure: *Path*

```
1: FindPath(t)
2: Add node t to array Path
3: if (t = Y) then
4:     return True
5: else if (t = N or t = NA or (⟨x_t⟩! = 0) and (⟨x_t⟩! = 1)) then
6:     return GreedyPath()
7: else if ⟨x_t⟩ = 0 then
8:     return FindPath(low(t))
9: else
10:    return FindPath(high(t))
11: end if
```

The FindPath algorithm traverses a BDD from top to bottom, one level at a time, recursively. Hence, the number of recursive calls to the algorithm is at most n and the algorithm's complexity is $O(n)$, where n is the height of the BDD. Note that n is equal to the number of variables in the BDD.

3.7.2 GreedyPath Algorithm

The *GreedyPath algorithm* finds a suboptimal path that leads to the Y terminal node. The GreedyPath algorithm does not guarantee an optimal solution, but many of its output paths are optimal.

Assume that $\langle x_i \rangle$ is a value assigned to the Boolean variable x_i by the Transmission Request Interpreter during the process of evaluating a transmission request. The notations high(t), low(t) and var(t) represent the high-child, the low-child and the variable of node t, respectively. The GreedyPath algorithm finds a suboptimal path that leads to the Y terminal node

and stores the new path in the array *Path*. The policy reasoner returns this array to the SSR as a set of opportunity constraints. The pseudo-code for the GreedyPath algorithm is given below.

Algorithm 2 GreedyPath

Require: BDD, p, $\langle x_i \rangle$
Ensure: *Path*

```
    GreedyPath (t)
 2: Add node t to Path
    if (t = Y) then
 4:     return Path
    else if (low(t) = N or low(t) = NA) then
 6:     GreedyPath(high(t))
    else if (high(t) = N or high(t) = NA) then
 8:     GreedyPath(low(t))
    else if (⟨x_t⟩ = 1) then
10:     GreedyPath(high(t))
    else if (⟨x_t⟩ = 0) then
12:     GreedyPath(low(t))
    else
14:     GreedyPath(high(t))
    end if
```

The Greedypath algorithm also traverses a BDD from the top to the bottom, one level at a time recursively. It finds a path towards the Y terminal node such that the value of each Boolean variable in the path is the same as its value assigned by the Transmission Request Interpreter unless this assigned value leads to the N terminal node. The algorithm complexity is $O(n)$, where n is the height of the BDD. Again as in the above case n is equal to the number of variables in the BDD.

An example of the Greedy path algorithm is given below:

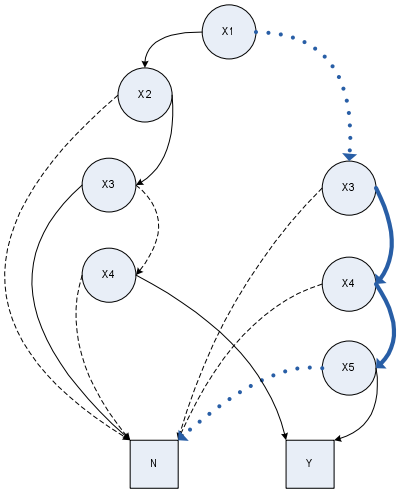

Figure 3.15: Greedy Path Algorithm

Consider the meta-policy MTBDD shown in the Figure 3.15. Let the values of the Boolean variables be as given below:

$x_1 : 0$

$x_2 : 0$

$x_3 : 1$

$x_4 : 1$

$x_5 : 0$

Depending on the values of the Boolean variables, the GreedyPath algorithm will follow the path shown in the figure till the node N, however, when it reaches the Node N, it backtracks one step to x_5 and flips the value of x_5 from 0 to 1, to reach the Y terminal. Thus the GreedyPath Algorithm backtracks once if it reaches the N terminal and finds a path to the Y terminal from that node.

3.7.3 Concept of Cost

When the transmission request is incomplete or is denied, the policy reasoner is supposed to return opportunity constraints. It would be of great value to the SSR if the opportunity constraints returned are Optimal. According to our analysis the optimality of the opportunity constraints depends of the two things given below:

1. Minimal changes to the original transmission request provided by the SSR.

2. The exact nature of the changes suggested by the Policy Reasoner.

The below example explains the meaning of *cost* or *Flip Cost* and emphasizes on its importance in returning constraints.

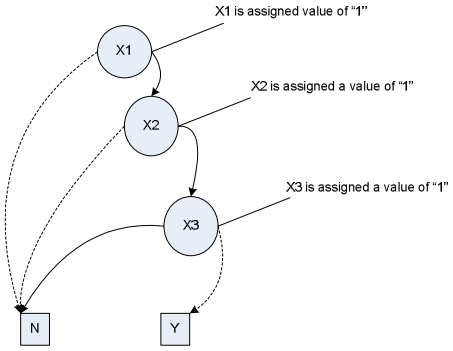

Figure 3.16: Concept of Cost

Consider the Meta-MTBDD shown in the Figure 3.16:

Assume the Boolean Encoding used was:

$x_1 : (Center\ Frequency = 235\ MHz)$

$x_2 : (Transmission\ Power \leq -10\ dbm)$

$x_3 : (Distance \leq 25\ miles$ from Location $72.35; 77.58; 0.0)$

Assume that the values assigned to the Boolean variables by the Transmission Request Interpreter are as below:

$x_1 : 1$

$x_2 : 1$

$x_3 : 1$

If the reasoner has to return a path to the "Y" terminal, the value of x_3 must be flipped from 1 to 0. Semantically this means that the Policy reasoner suggests the Radio to change its location. This suggestion though correct is not feasible for the radio. As a result of this we introduced the concept of cost. The parameters which the radio can manipulate easily such as Center Frequency, Transmission Power, etc. are given a lower cost, while the parameters which the radio cannot manipulate relatively easily are given higher cost.

We assume that the SSR calculates the cost for each parameter depending on the complexity involved in changing the values of these parameters and sends over these costs to the PR along with the Transmission request.

At this point we modify the definition of the transmission request given in section 3.6. The PR assumes that the transmission request submitted by the SSR has a structure as defined below:

Definition 2 *Let $a_1, a_2, ..., a_k$ represent a sequence of attribute names, $v_1, v_2, ..., v_k$ represent a sequence of values and $c_1, c_2, ..., c_k$ represent a sequence of costs where v_i is an value of*

the attribute a_i and c_i is the cost of the attribute a_i. The transmission request is defined as $r = \{(a_1, v_1, c_1), (a_2, v_2, c_2), \cdots, (a_k, v_k, c_k)\}$.

The Transmission Request Interpreter parses the request and analyzes it semantically. It then assigns values and "flip cost" to each of the Boolean Variables. These flip costs can then be used by the MinCostPath and the AllMinCostPath algorithms to return optimal opportunity constraints to the SSR.

3.7.4 MinCostPath Algorithm

The *MinCostPath algorithm* is used by the reasoner when it needs to find the *optimal* opportunity constraints in response to an invalid or incomplete transmission request sent by the System Strategy Reasoner. The MinCostPath algorithm finds an optimal opportunity constraint by solving the following problem.

Problem 1 *Suppose that $X = (x_1, \ldots, x_n)$ is a sequence of BDD variables and that each variable, x_i, is associated with a constant non-negative cost c_i. In the BDD, find a path from the root node to the Y-terminal node such that the cost sum, $\alpha_1 c_1 + \ldots + \alpha_n c_n$, is minimized, where α_i denotes a Boolean value assigned to x_i.*

In the MinCostPath algorithm, we assume that the variable ordering is $x_1 < \ldots < x_n$ and the nodes are numbered from the bottom of the BDD, i.e., if $var(x)$ represents the Boolean variable of node x then $var(i) > var(j) \Rightarrow i < j$. We define $Size(B)$ as the number of nodes in BDD B. We define the following arrays used in the algorithm:

- $c[i]$= The flip cost of the Boolean variable x_i.
- $MC[i]$= The minimum cost sum of any path between node i and the Y terminal node.

- $Path[i]$= The Boolean value for node i such that the cost sum of the path that the node is on is minimized.

- $\langle x_i \rangle$ = The value assigned to the Boolean variable x_i by the Transmission Request Interpreter during the process of evaluating a transmission request.

The pseudo-code of the algorithm is given below:

Algorithm 3 MinCostPath

Require: BDD, $c[i]$
Ensure: $Path$
$MW[0] = +\infty$
$MW[1] = 0$
for $i = 2$ to $Size(B) - 1$ **do**
  $l = low(i)$
  $h = high(i)$
  $v = var(i)$
  if $\langle x_v \rangle = 0$ **then**
    if $(MC[l] \leq MC[h] + c[v])$ **then**
      $MC[i] = MC[l]$
      $Path[i] = 0$
    else
      $MC[i] = MC[h] + c[v]$
      $Path[i] = 1$
    end if
  else
    if $(MC[h] \leq MC[l] + c[v])$ **then**
      $MC[i] = MC[h]$
      $Path[i] = 1$
    else
      $MC[i] = MC[l] + c[v]$
      $Path[i] = 0$
    end if
  end if
end for

Explanation: In order to avoid a path that terminates at the N terminal node we set $MC[N]$ $= +\infty$ and $MC[Y] = 0$. The cost from each node to the Y terminal is then calculated. The cost calculation starts from the bottom nodes of the meta-policy BDD and proceeds towards the root. The MinCostPath algorithm, guarantees that the path it returns is the most optimum path from the root node to the Y terminal node.

Consider the below meta-policy BDD shown in the Figure 3.17.

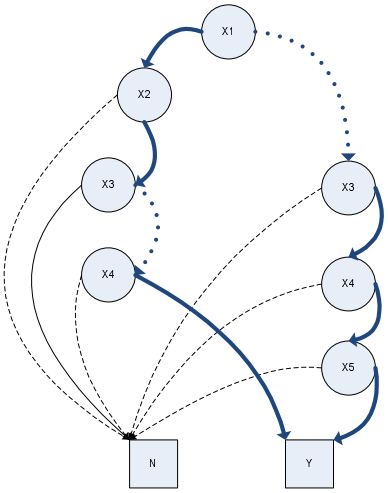

Figure 3.17: MinCostPath Algorithm

Let the values of the Boolean variables in the Meta-MTBDD be:

$x_1 : 0$

$x_2 : 0$

$x_3 : 1$

$x_4 : 1$

$x_5 : 0$

Two paths from the root node to the terminal node have been marked in the below figure.

One of the paths requires flipping the value of x_1 while the second path involves flipping the value of x_5. The MinCostPath algorithm will evaluate the costs for each path from the root node to the Y node and then return the path with the minimum cost.

3.7.5 AllMinCostPath Algorithm

The *AllMinCostPath* algorithm is used by the reasoner to find multiple optimal opportunity constraints. This happens when the Meta-policy BDD has multiple paths with the same minimum cost. An simple illustration of this is as under. Consider the below given two policies:

- *Policy* 1 : Transmit if Center frequency is 235 MHz OR 240 MHz OR 245 MHz.
- *Policy* 2 : Transmit if *Operation Mode* = *Special Event* AND TxmPower $\leq$ 20 *dbm* AND CenterFrequecy is 270 MHz.

Now suppose the cost or flipping the Boolean Variables are as under:

Center Frequency: 1 unit

Operation Mode: 20 units

Transmit Power: 3 units

Let us assume that the current state of the device and the transmission request parameters violate both the policies and the transmission has to be denied. In this case, one of the reasoning algorithms will be used to send back the opportunity constraints to the SSR. In this particular example, there are three optimal solutions that can be returned back to

the SSR. The MinCostPath algorithm returns any one of these, while the AllMinCostpath algorithm, "ORs" all the solutions and sends back the reply. If the MinCostPath algorithm is used it may send back an opportunity constraint saying, "Change the Center Frequency to 235 MHz", while if the AllMinCostPath algorithm is used, it will send back an opportunity constraint saying, "Change the Center Frequency to 235 MHz or 240 MHz or 245 MHz". Such reply is always beneficial for the System Strategy Reasoner as it gets multiple sets of opportunity constraints to choose from rather than just one set.

The AllMinCostPath algorithm is capable of returning all the minimum cost paths to the System Strategy Reasoner. However, in some cases the number of optimal paths in the meta-policy BDD can be large and this may affect the response time of the reasoner. Hence the AllMinCostPath algorithm expects a parameter from the SSR which will inform it how many optimal solutions the SSR expects.

In the AllMinCostPath algorithm, we assume that the variable ordering is $x_1 < ... < x_n$ and the nodes are numbered from the bottom of the BDD, i.e., if $var(x)$ represents the Boolean variable of node x then $var(i) > var(j) \Rightarrow i < j$. The algorithm is divided into two parts as follows:

1. CreateCostMap - In this part the cost from each node to the "Y" terminal is calculated. Also the total number of minimum cost paths is calculated in this part of the algorithm.

2. AllPathTraversal - This part is responsible for iteration through the meta-policy BDD

for as many numbers of times as many minimum cost paths exist or the number of optimal solution the SSR expects, whichever is smaller. Each Iteration results in one path. In order that the same path is not visited twice, a visited path is blocked by changing its weight to infinity before the next iteration.

Pseudo Code for these two parts of the algorithm is given below. Algorithm 4 gives the pseudo code for CreateCostMap part while Algorithm 5 gives the pseudo code for the All-PathTraversal part. We define Size(BDD), as the number of nodes in BDD B. We define the following arrays and terms used in the pseudo code:

- $c[i]$= The flip cost of the Boolean variable x_i.
- $MC[i]$= The minimum cost sum of any path between node i and the Y terminal node.
- $\langle x_i \rangle$ = The value assigned to the Boolean variable x_i by the Transmission request Interpreter during the process of evaluating transmission request.
- $MaxPaths$= The maximum number of optimal solutions that SSR expects from the PR

Algorithm 4 CreateCostMap

Require: BDD, $c[i]$
Ensure: $MC[i]$,$EQNodeList[]$

```
    MC[0] = +∞
 2: MC[1] = +∞
    MC[2] = 0
 4: for (i = 3 to Size(B) − 1) do
        l = low(i)
 6:     h = high(i)
        v = var(i)
 8:     if ⟨x_v⟩ = 1 then
            MC[i] = Min(MC[l] + c[v], MC[h])
10:         if (MC[l] + c[v] == MC[h]) then
                EQNodeList.add(⟨x_v⟩)
12:         end if
        else
14:         MC[i] = Min(MC[l], MC[h] + c[v])
            if MC[l] == MC[h] + c[v] then
16:             EQNodeList.add(⟨x_v⟩)
            end if
18:     end if
    end for
```

Algorithm 5 AllPathTraversal

Require: $MC[i]$, $EQNodeList[]$,$c[i]$, $RefRootNode$,$MinCostPathNum$,$MaxPaths$
Ensure: $Path[]$

```
    if MaxPaths < MinCostPathNum then
        MinCostPathNum = MaxPaths
 3: end if
    for (i = 0 to MinCostPathNum) do
        rootNode = refRootNode
 6:     l = low(i)
        h = high(i)
        v = var(i)
 9:     while (rootNode! = Terminalnode) do
            if (⟨x_v⟩ = 1) then
                if (MC[l] + c[v] < MC[h]) then
12:                 rootNode = low(rootNode)
                else if (MC[l] + c[v] == MC[h]) then
                    rootNode = high(rootNode)
15:                 RemNodeList.add(⟨x_v⟩)
                else
                    rootNode = high(rootNode)
18:             end if
            else
                if (MC(h) + c[v] < MC[l]) then
21:                 rootNode = high(rootNode)
                else if (MC[h] + c[v] == MC[l]) then
                    rootNode = low(rootNode)
24:                 RemNodeList.add(⟨x_v⟩)
                else
                    rootNode = low(rootNode)
27:             end if
            end if
            if (rootNode == Terminalnode) then
30:             if (RemNodeListhaselements) then
                    j = GetLastNodeFromList(RemNodeList)
                    MC[high(high(j))] = +∞
33:                 MC[low(high(j))] = +∞
                end if
            end if
36:     end while
    end for
```

3.8 BDD-based Reasoner

The reasoner explained in the Section 3.1 through Section 3.7 is an MTBDD-based reasoner. In the MTBDD-based reasoner policies are first represented as BDDs. These BDDs are then combined to get a meta-policy MTBDD which is then processed upon and converted into a meta-policy BDD. The meta-policy BDD is then used for reasoning. The Approach in the MTBDD-based reasoner is illustrated in the Figure 3.19

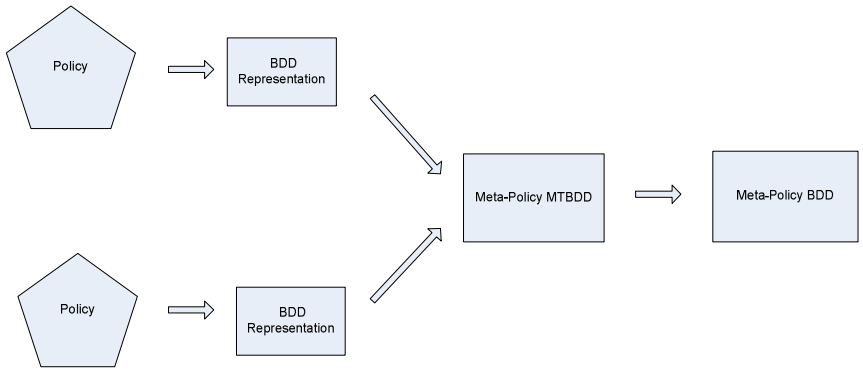

Figure 3.18: MTBDD-based Reasoner Approach

The intermediate MTBDDs that MTBDD-based reasoner uses for combining the policies may have upto three terminal nodes viz. N, NA and Y. The terminal node combination approach that MTBDD-based reasoner uses while combining the policies is shown below:

$Allow(Y) \wedge Allow(Y) \rightarrow Allow(Y)$

$Allow(Y) \wedge Not\ Applicable(NA) \rightarrow Allow(Y)$

$Disallow(N) \wedge Not\ Applicable(NA) \rightarrow Disallow(N)$

$Disallow(N) \wedge Disallow(N) \rightarrow Disallow(N)$

$Allow(Y) \wedge Disallow(N) \rightarrow Disallow(N)$

The result of using this terminal node combination approach is that, effectively, in case of two permissive policies we "OR" the two policies and in case of two prohibitive policies or one permissive and one prohibitive policy we "AND" the two policies.

This knowledge leads us to an intuitive approach, wherein, we can modify the specified terminal combination approach to use just BDDs instead of MTBDDs while combining any two policies. The Approach used in the BDD-based reasoner is illustrated in the Figure **??**.

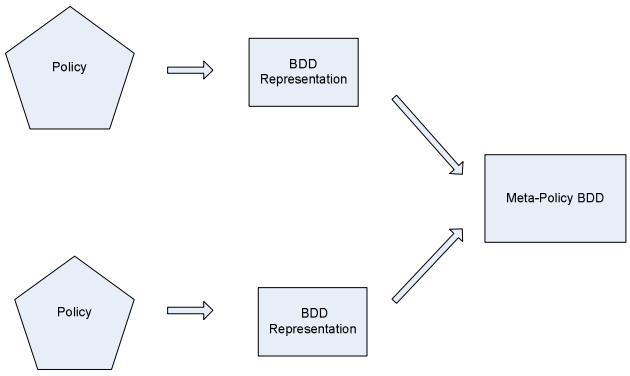

Figure 3.19: BDD-based Reasoner Approach

The Terminal node combination approach used in the BDD-based reasoner would be as under:

Two permissive policies are always "ORed" together.

$Y \vee Y \rightarrow Y$

$Y \vee N \rightarrow Y$

$N \vee N \rightarrow N$

One permissive Policy and one prohibitive policy or two prohibitive policies are always "ANDed" together.

$Y \wedge Y \rightarrow Y$

$Y \wedge N \rightarrow N$

$N \wedge N \rightarrow N$

The example in Figure 3.20 shows the meta-policy BDDs when a permissive policy and a prohibitive policy are combined using the MTBDD-based reasoner and the BDD-based reasoner. We can see that, in both cases the final meta-policy BDD is the same.

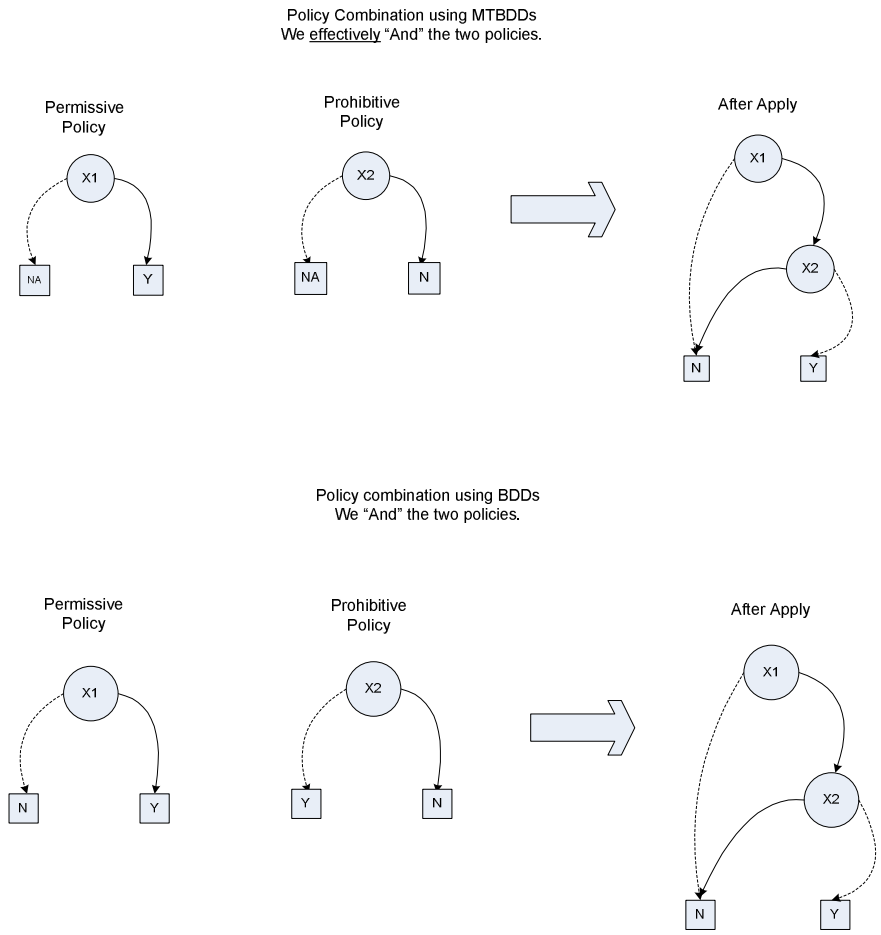

Figure 3.20: Merging a permissive and prohibitive policy

The example in Figure 3.21 shows the meta-policy BDDs when two permissive policies are combined using the MTBDD reasoner and the BDD reasoner. We can see that, in both cases the final meta-policy BDD is the same.

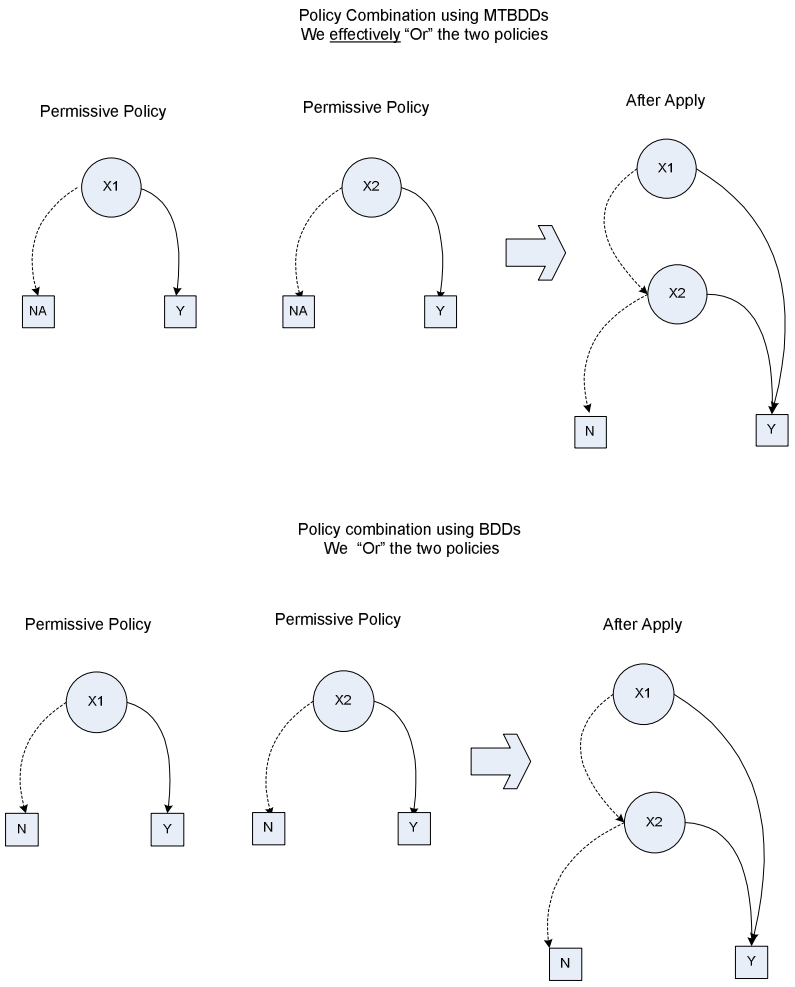

Figure 3.21: Merging two permissive policies

However, there is a slight complication in the procedure to combine the permissive and prohibitive policies. Consider the combination of a permissive and prohibitive policy using the BDD-based reasoner in Figure 3.20. In this combination, when we merge the "N" terminal of a permissive and "Y" terminal of the Prohibitive Policy. The correct result here would an "NA" terminal. However, a BDD does not have an "NA" terminal and hence this policy combination is incorrect as it would lead to loss of valid paths if any other permissive

policy is combined with this meta-policy.

In order to avoid this loss of valid paths in the BDD based reasoner, we change the way in which the policies are combined. We maintain two meta-policy BDDs one for all the permissive policies and a second for all the prohibitive policies. Any new permissive policy is combined with the permissive meta-policy BDD while any new prohibitive policy is combined with the prohibitive meta-policy BDD. When all the policies are combined with their respective BDDs, the permissive and prohibitive BDDs are combined. Thus, in this approach, we combine the permissive and the prohibitive policy BDDs only at the last instant. At this point, the NA terminal is similar to the N terminal as no more policies need to be merged with the meta-policy BDD. This combination approach is advantageous as it leads to the generation of significantly less number of nodes as compared to the nodes generated in the intermediate meta-policy MTBDD in the MTBDD-based reasoner thus reducing the memory requirements.

Note that the current implementation of BDD reasoner is achievable, only because we use separate permissive and prohibitive policies. However, in the future, it is possible to have spectrum access policies which have both the permissive and prohibitive rules in the same policy. As explained in Section 3.3, permissive or prohibitive policies require two terminal nodes each for representatio. However, if we have a combined policy which has permissive as well as prohibitive rules, then, such a policy will require three terminal nodes for its representation. This is because, such a policy can have all the three states viz. “Allow Transmission”, “Deny Transmission” and “Not Applicable” as possible results. Such policies

cannot be represented using BDDs and hence the BDD reasoner cannot be used "as is" in this case. In this case we will need additional processing in the policy parser module so that we can separate the permissive and prohibitive rules, and create separate BDDs for each of these rules so that they can be combined with the meta-policy BDD.

Chapter 4

Implementation Results

We have developed prototype for a device independent policy reasoner in Java that facilitates Dynamic spectrum access. For the purpose of experimental evaluation we divided the policy reasoner components into two sections as shown in Figure 4.1.

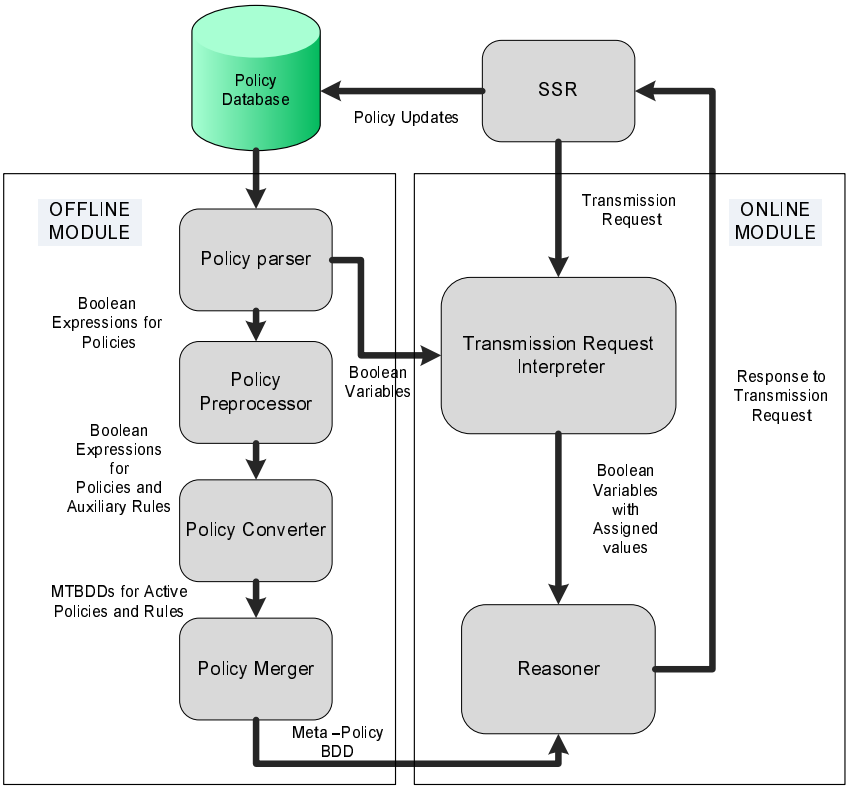

Figure 4.1: BRESAP Implementation

These two sections and the corresponding processing tasks are as described below:

- Offline Section: This section contains the modules which are responsible for parsing of policies and creating the meta-policy BDD. It consists of the Policy Parser, Policy Preprocessor, Policy converter and Policy Merger modules. The processing in the offline section can be carried out while initializing the radio or whenever the radio downloads a new policy and hence we consider the processing involved in this module to be less time critical.

- Online Section: This section consists of the Transmission Request Interpreter and the Policy Reasoner modules. When the policy engine gets a transmission request, it needs to evaluate the request and send back an appropriate reply to the System Strategy Reasoner. This function of the reasoner is time critical and thus in general, it is the performance of the online section that determines BRESAP performance.

For the purpose of evaluation we generated SWRL policies such as Listen Before Talk (LBT) and other time and radio location dependent policies. The number of policies was varied from five to fifty while varying the number of *Atomic Boolean Expressions* (*AEs*) in each of the policies from 5 to 15. The experiments were conducted on $Intel\ Core\ 2\ CPU$ 1.86 GHz machine with 3 $GB\ RAM$.

4.1 Limitations of BRESAP

BRESAP currently has some practical limitations. This is due to the fact that the build algorithm in the policy converter module has exponential complexity. As a result of this current implementation of BRESAP is limited to handle policies up to $22 - 24$ boolean variables. Each variable added after this increases the time required for execution. However, this is only an issue with the current BRESAP implementation. There are C++ packages like CUDD [31], which are available and have been optimized for MTBDD implementations. In [27], authors have successfully used the CUDD package to represent xml-based policies having upto 50 boolean variables as MTBDDs.

As explained in Section 3.5, the preprocessor module creates auxiliary rules, which when combined with the Meta-MTBDD, remove the illegal paths from the Meta-MTBDD. In the process of creating auxiliary rules, the preprocessor module combines variables from different policies. This in many cases, results in auxiliary rules with more than twenty variables. As a result, in some cases, the offline module took a long time for execution. In order to avoid this, the performance evaluation was divided into two parts, viz.

1. Results without the Preprocessor Module.

2. Results with the Preprocessor Module.

4.2 Results without the Preprocessor Module

In this set of results, the policies were created manually using a combination of "and" and "or" such that all the policies had distinct variables. This is not a realistic scenario; however, this testing was done to test the online and the offline performance of the reasoner for a large number of boolean variables. In this case, the number of variables in each of the policies was varied from 5 to 15 while the number of policies was varied from 5 to 50.

4.2.1 Offline Module Performance

Figure 4.2 shows the performance of the offline section, for a set of 30 policies, in a semi-logarithmic format. These results show the time taken by the offline section to parse a policy into a Boolean expression, build a BDD from the Boolean expression and then combine

this BDD with the existing Meta-MTBDD. The blue curve with triangle markers shows the average time taken by the policy parser module to parse a policy when the number of Boolean variables or Atomic Boolean Expressions (AEs) is varied from 5 to 15; the green curve with circle markers shows the average cumulative time taken for the generation of the BDD. The red curve with square markers shows the average cumulative time taken for the combination of the BDD with the existing Meta-MTBDD. This figure also helps us evaluate the relative performance of the components of the offline section. We can see from the figure that the policy parsing and meta-policy MTBDD generation are polynomial time operations while the BDD generation from the Boolean expressions is exponential in time.

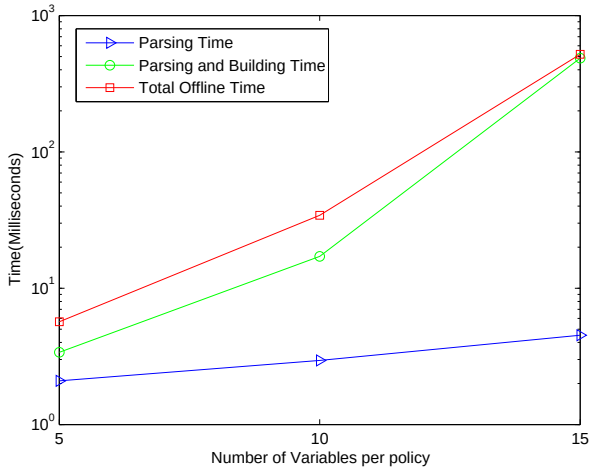

Figure 4.2: Offline processing time vs. number of AEs

We further analyzed the performance of the Policy Merger module. The Figure 4.3 shows the time taken to generate the Meta-MTBDD when the number of policies is varied from 5 to 50 and the number of AEs in each of the policies is varied from 5 to 15.

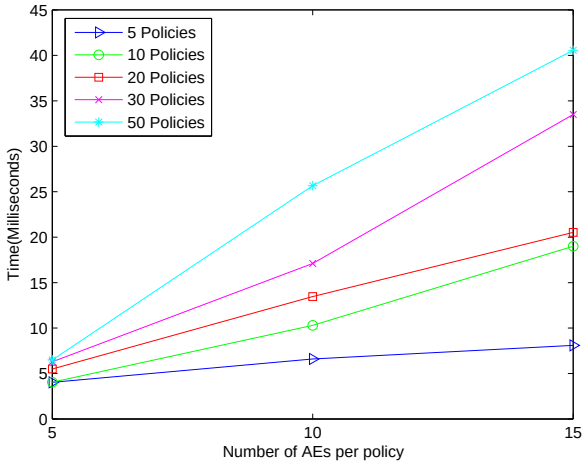

Figure 4.3: Policy Merging time vs. number of AEs

4.2.2 Online Module Performance

The Online section consists of the Transmission Request Interpreter module and the Policy Reasoner module.

4.2.2.1 Transmission Request Interpreter Performance

The Transmission Request Interpreter Module is responsible for parsing the transmission request and interpreting it. Depending on values of the transmission request parameters and the data in the transmission request this module assigns appropriate Boolean values and flip costs to the AEs in the meta-policy BDD, which helps the reasoner to evaluate the request. Figure 4.4 shows that the performance of transmission request interpreter module is linearly proportional to the number of AEs.

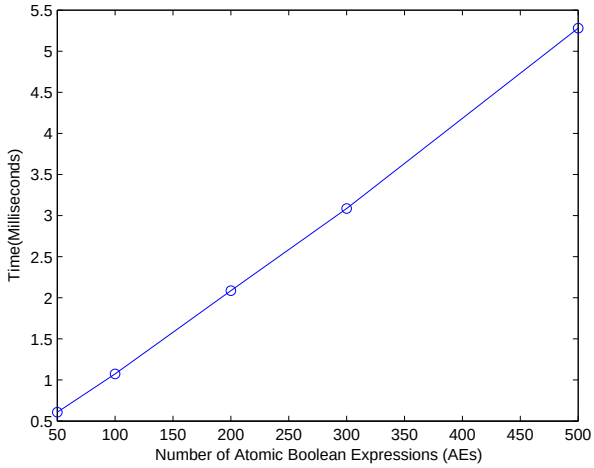

Figure 4.4: Number of nodes in meta-policy MTBDD vs Parsing Time

4.2.2.2 Policy Reasoner Performance

The time taken for reasoning and returning the constraints depends on the number of AEs in the policies and the type and complexity of the policies that have been used to create the Meta-policy BDD. The policy reasoner traverses the meta-policy BDD using the values that the Transmission Request Interpreter module has assigned to the Boolean Variables. The reasoner first uses the FindPath algorithm for this purpose. If the transmission request is allowed it sends back a positive reply to the System Strategy Reasoner, however if the transmission request is denied or if the transmission request is incomplete then the reasoner determines the parameters that if changed would allow the transmission and returns back these constraints. The reasoner may use any one of the three algorithms to return constraints viz. GreedyPath algorithm, MinCostPath Algorithm or AllMinCostPath Algorithm. In general, as the number of policies increases and as the number of boolean variables or constraints

per policy increases the time taken for reasoning increases.

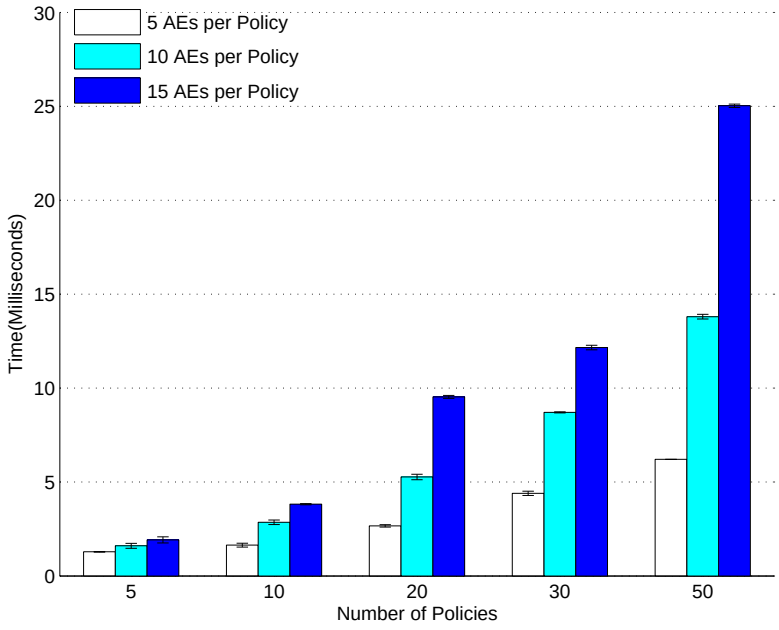

Figure 4.5: Total Online Processing Time for MinCostPath Algorithm

The Figure 4.5, shows the total time taken by the policy reasoner to return constraints as the number of policies are increased from 5 to 50 and the number of variables per policy are increased from 5 to 15. The reasoning algorithm used in this case was MinCostPath Algorithm. The total time plotted on the Y axis includes, the time taken by the Transmission Request Interpreter, the time taken by the FindPath Algorithm to check if the Transmission Request is allowed or denied and the time taken by the MinCostPath algorithm in case the transmission request is denied. Note that the results in Figure 4.5 only represent the cases where the transmission request was denied and the MinCostPath algorithm was used to return the constraints.

4.2.2.3 Algorithm Performance Comparison

As explained in Section 3.7, the time complexity of the GreedyPath algorithm is $O(n)$, where n is the number distinct boolean variables in the meta-policy BDD, while the time complexity of the MinCostPath algorithm is $O(m+n)$ where m is the number of nodes in the meta-policy BDD while n is the number of distinct boolean variables in the meta-policy BDD. The time complexity of the AllMinCostPath algorithm is $O(m+np)$ where m is the number of nodes in the meta-policy BDD , n is the number of nodes in the path while p is the number of Minimum cost paths in the meta-policy BDD. In our experiments with the AllMinCostPath algorithm, the maximum number of minimum cost paths returned was restrained to "3", as we assumed that the SSR sets this parameter. The Figure 4.6 shows a performance comparison of the three algorithms for a set of 50 policies as the number of boolean variable per policy is increased from 5 to 15. Note that this figure only shows the time taken by the reasoning algorithms after the transmission request has been denied by the Policy Reasoner using the FindPath Algorithm.

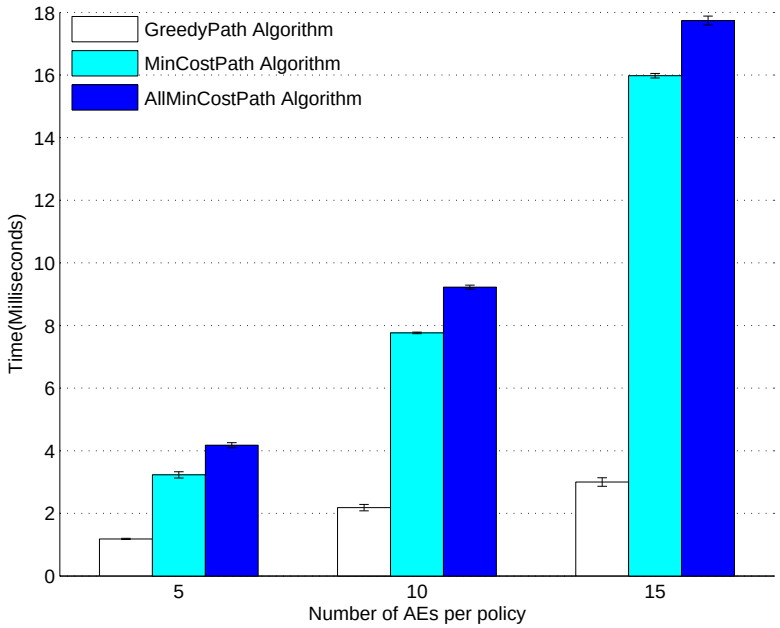

Figure 4.6: Algorithm Comparison

As expected the time taken by the GreedyPath algorithm is much less as compared to the time taken by the MinCostPath and the AllMinCostPath algorithms. Also the time taken by the AllMinCostPath algorithm to return the constraints, is more than the time taken by MinCostpath Algorithm as the AllMinCostpath algorithm iterates the meta-policy BDD multiple times while the MinCostPath algorithm iterates through the meta-policy BDD just once.

4.3 Results with the Preprocessor Module

In this set of results, the policies were realistic and similar to the policies given in [1]. Each of the policies had on an average 5 variables and the number of policies was varied from 2 to 15 and the performance of the offline and online modules was tested. The results in the Section

4.2 are without the preprocessor module. In this section we give results with the preprocessor module in place. The performance of the online module using the AllMinCostPath algorithm can be seen in the Figure 4.7.

4.3.1 Online Module Performance

The Figure 4.7, shows the total time taken by the online module of the policy reasoner to return constraints as the number of policies are increased from 2 to 15 with an average of 5 variables per policy. The reasoning algorithm used in this case was MinCostPath Algorithm. The time plotted on the Y axis includes, the time taken by the Transmission Request Interpreter, the time taken by the FindPath Algorithm to check if the Transmission Request is allowed or denied and the time taken by the MinCostPath algorithm in case the transmission request is denied. Note that the results in Figure 4.7 only represent the cases where the transmission request was denied and the MinCostPath algorithm was used to return the constraints.

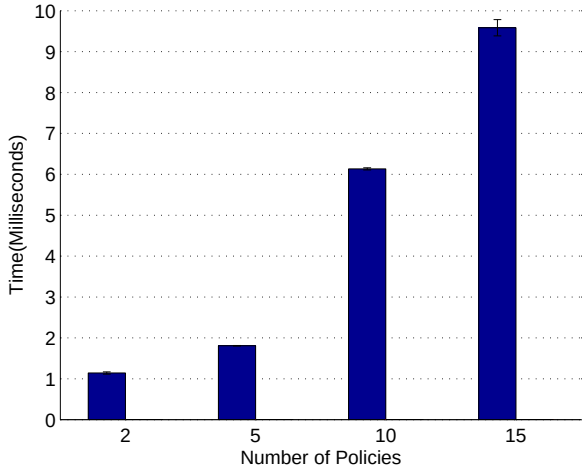

Figure 4.7: Online Processing Time Vs Number of Policies

The Figure 4.8 shows a performance comparison of the three algorithms as the number of policies was increased from 2 to 15 with on an average 5 boolean variables or AEs per policy. Note that this figure only shows the time taken by the reasoning algorithms after the transmission request has been denied by the Policy Reasoner using the FindPath Algorithm.

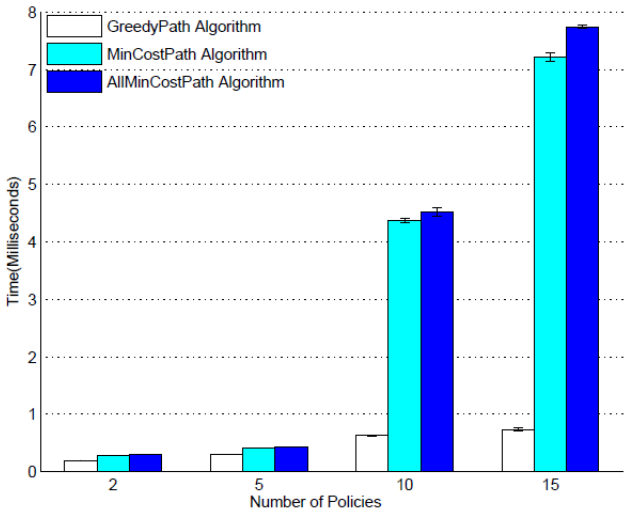

Figure 4.8: Algorithm Comparison

In this set of results, the AllMinCostPath algorithm returned *one* set of opportunity constraints when 2,5 and 10 policies were used, while it returned two sets of opportunity constraints in the case when 15 policies were used. This is the reason for the AllMinCostPath algorithm taking more time as compared to that of the MinCostPath algorithm in the case of 15 policies.

4.3.2 Memory

We studied the memory requirements of BRESAP. As the number of policies increases, the number of nodes required to represent the meta-policy BDD goes on increasing and hence the memory required goes on increasing. Figure 4.9, shows the results.

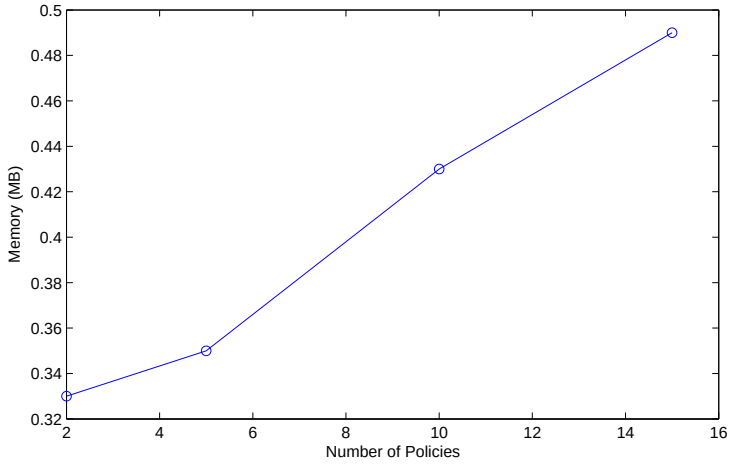

Figure 4.9: Memory Requirement Vs Number of Policies

4.4 BDD-based Reasoner Evaluation

As explained in Section 3.8, the BDD-based reasoner, does not use an intermediate MTBDD for the generation of the final meta-policy BDD. Though, the intermediate MTBDD generated in the MTBDD-based reasoner has significantly more nodes as compared to the nodes generated in the BDD-based reasoner, the final meta-policy BDDs are the same in both the cases. As a result of this, the online performance for both BDD-based reasoner and the MTBDD-based reasoner is the same.

In the offline section, the BDD-based reasoner does not need post processing of any intermediate form and as a result, the offline time taken by the BDD-based reasoner is slightly less as compared to the MTBDD-based reasoner.

In order to compare the BDD-based reasoner with the MTBDD-based reasoner, we compare the maximum number of nodes generated in the MTBDD-based reasoner and the BDD-based reasoner while combining the policies. Figure 4.10 shows the results for the same.

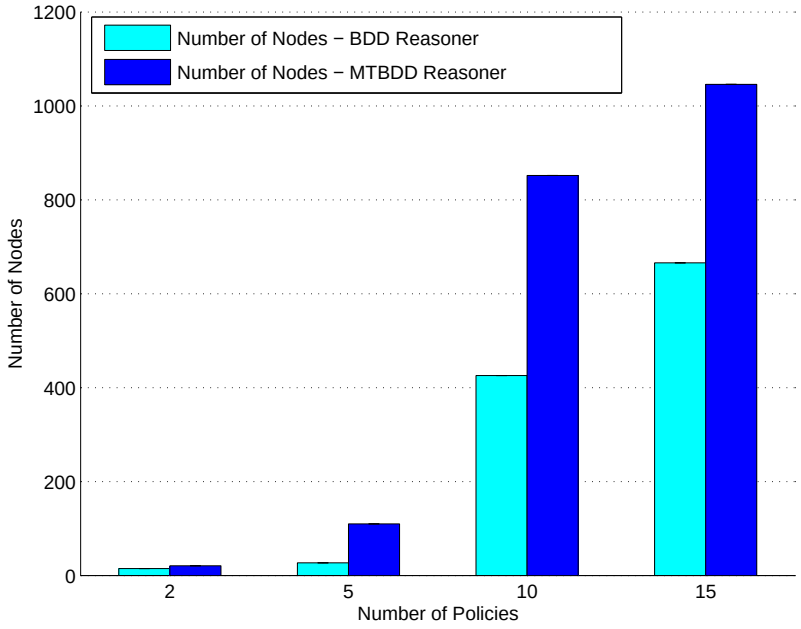

Figure 4.10: Number of Nodes Vs Number of Policies

As seen from the figure the maximum number of nodes generated in the case of the BDD-based reasoner is lesser as compared to the maximum nodes generated in the MTBDD-based reasoner and thus the BDD-based reasoner has reduced memory requirements as compared to the MTBDD-based reasoner.

Chapter 5

Conclusion and Future Work

In this Thesis, we described the design and implementation of a novel policy reasoner. The proposed policy reasoner uses Binary Decision Diagrams (BDDs) and Multi-Terminal Binary Decision Diagrams (MTBDDs) to represent, interpret and process policies in an efficient manner. It uses a set of efficient graph-theoretic algorithms to translate policies into BDDs, merge policies into a single meta-policy, and compute opportunity constraints in case of a denied transmission request. The proposed policy reasoner has the capability to process either under-specified or invalid transmission requests sent by the System Strategy Reasoner. We have also proposed three different graph-based algorithms for computing the opportunity constraints. This work takes a small step forward in addressing the problem of designing policy reasoners that can correctly process complex policies while satisfying the stringent spectrum access control requirements of the spectrum stakeholders.

Several promising directions exist for future work. First we plan to collaborate with the other ongoing research on policies so that we get a chance to work with more realistic

spectrum access policies. Secondly we would like to Integrate our work with Cognitive Radio Open Source System (CROSS) [32] which is a research project at wireless@VT [35]. CROSS is an distributed and modular cognitive radio system that provides portability and interoperability between components developed in different programming languages, across different platforms. We plan to design and implement an communication API which will enable the interaction between BRESAP and other components in CROSS. Also we are interested in exploring some of the optimized BDD packages available to build the policy reasoner which we feel will definitely improve the performance of our reasoner. Finally we plan to exploit ontologies and ontological reasoning to deal with the cases in which policies can refer to radios, radio capabilities and parameters and the relevant properties of the radio environment that are defined using ontologies.

Bibliography

[1] "SWRL FOL Policies." http://xg.csl.sri.com/policies.php

[2] Qing Zhao, "A survey of dynamic spectrum access: signal processing, networking, and regulatory policy," IEEE Signal Processing Magazine 2007.

[3] A. E. Leu, M. McHenry, and B. L. Mark, "Modeling and analysis of interference in Listen-Before-Talk spectrum access schemes," International Journal of Network Management, vol. 16, no. 2,pp. 131-147, 2006.

[4] D. Elenius, G. Denker, M.O. Stehr, R. Senanayake, C. Talcott, D. Wilkins, "CoRaL - Policy Language and Reasoning Techniques for Spectrum Policies," Policies for Distributed Systems and Networks, 2007. POLICY'07, 13-15 June 2007, pp. 261-265

[5] F. Perich, "Policy-based network management for NeXt generation spectrum access control," DySPAN2007, 17-20 April 2007, pp: 496-506

[6] F. Perich, M. McHenry, "Policy-based spectrum access control for dynamic spectrum access network radios," Web Semantics: Science, Services and Agents on the World Wide Web, 2008.

[7] D. Wilkins, G. Denker, M.O. Stehr, D. Elenius, and R. Senanayake, "Policy-based cognitive radios," IEEE Wireless Communications Magazine, vol 14, 2007, pp.41-46.

[8] H. R. Andersen, "An Introduction to Binary Decision Diagrams," Lecture Notes, 1999, IT University of Copenhagen.

[9] DARPA XG Working Group, "The XG Vision, Request for Comments," prepared by BBN Technologies, July 2003.

[10] H. R. Andersen, "An Introduction to Binary Decision Diagrams," Lecture Notes, 1999, IT University of Copenhagen.

[11] D. Elenius, G. Denker, and D. Wilkins, "XG Policy Architecture. Request for comments," SRI International, Tech. Rep., 2007.

[12] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "OWL Web Ontology Language Reference," W3C Recommendation, February 2004.

[13] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M.Dean, "SWRL: A semantic web rule language combining OWL and RuleML," W3C Member Submission, 21 May 2004.

[14] M. Fujita, P. C. McGeer, and J. C.-Y. Yang, "Multi-terminal binary decision diagrams: An efficient datastructure for matrix representation," Formal Methods in System Design, vol 10, 1997, pp. 149-169.

[15] M. Sauerhoff, I. Wegener, and R. Werchner, "Optimal Ordered Binary Decision Diagrams for Fanout-free Circuits," in Proc. SASIMI96, 1996, pp. 197-204.

[16] DARPA XG Working Group, "XG Policy Language Framework. Request for Comments," Version 1.0, Prepared by BBN Technologies, Cambridge MA, USA, April 2004.

[17] K. Feeney, D. Lewis, P. Argyroudis, K. Nolan, D. OSullivan, "Grouping Abstraction and Authority Control in Policy-based Spectrum Management," DySPAN2007, 17-20 April 2007, pp.: 363-371

[18] D. Lewis, K. Feeney, K. Foley, L. Doyle, T. Forde, P. Argyroudis, J. Keeney, D. OSullivan, "Managing Policies for Dynamic Spectrum Access", 1st International IFIP TC6 Conference on Autonomic Networking, AN2006, Sept. 2006, pp.: 285 297

[19] E. Lupu and M. Sloman, "Conflicts in Policy-Based Distributed Systems Management", IEEE Transactions on Software Engineering, Vol. 25, No. 6:852-869, 1999

[20] E. Lupu, M. Sloman, "Conflict Analysis for Management Policies," Proceedings of the fifth IFIP/IEEE international symposium on Integrated network management 1997, pp: 430 443

[21] M.Charalambides, P.Flegkas, G. Pavlou, A.K.Bandara,E.C. Lupu, A.Russo, N. Dulav, M. Sloman, J.R. Loyola, "Policy conflict analysis for quality of service management," Workshop on Policies for Distributed Systems and Networks, (POLICY2005), 6-8 June 2005, pp: 99- 108

[22] M.Charalambides, P.Flegkas, G. Pavlou, J.R. Loyola, A.K.Bandara, E.C. Lupu, M. Sloman, N. Dulav, "Dynamic Policy Analysis and Conflict Resolution for DiffServ Quality of Service Management," Network Operations and Management Symposium, 2006. NOMS 2006,pp: 294 - 304

[23] P. Flegkas, P. Trimintzios, and G. Pavlou, "A Policy-based Quality of Service Management System for IP DiffServ Networks," IEEE Network Magazine, vol. 16 No. 2, pp. 50-56, 2002.

[24] H.Hamed., E.Al-Shaer, "Taxonomy of Conflicts in Network Security Policies," IEEE Communications Magazine, Vol. 44, March 2006, pp: 134 - 141

[25] E. Al-Shaer and H. Hamed, "Modeling and Management of Firewall Policies," in IEEE Transactions on Network and Service Management (eTNSM 2004), Volume 1-1, April 2004.

[26] G. Denker, D. Elenius, R. Senanayake, M.-O. Stehr, and D. Wilkins, "A policy engine for spectrum sharing," in Proc. of 2007 IEEE DySPAN, Apr. 2007, pp. 55-65.

[27] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. "Verification and change-impact analysis of access-control policies," In Proceedings of the 27th International Conference on Software Engineering (ICSE), 2005, pp. 196-205.

[28] P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo, "An Algebra for Fine-Grained Integration of XACML Policies," 14th ACM symposium on Access control models and technologies, June 2009, pp. 63-72.

[29] G. Kolaczek. "Specification and verification of constraints in role based access control for enterprise security system," In International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003, pp. 190-195.

[30] D. Jackson, "Automating first-order relational logic," In ACM SIGSOFT International Symposium on the Foundations of Software Engineering, Nov. 2000, pp. 130-139.

[31] F. Somenzi. CUDD: The CU decision diagram package. http://vlsi.colorado.edu/ fabio/CUDD/.

[32] "CROSS." http://cornet.wireless.vt.edu/trac/wiki/Cross

[33] "SWRL FOL." http://www.daml.org/2004/11/fol/

[34] "JDOM." http://www.jdom.org/

[35] "Wireless @ Virginia Tech." http://wireless.vt.edu/