CHAPTER 5

PARALLEL CLASSIFIER USING RIPPER

5.1 INTRODUCTION

Denial of Service, in general, is a more expensive attack on a communication infrastructure supporting time sensitive applications such as real-time systems, as it reduces the bandwidth and renders the application to stop abruptly resulting in unrecoverable damage to the system. This chapter presents a knowledge based intrusion detection model building and sequential and parallel implementation of the model for attack detection against the network and information infrastructure implementing the real-time system. The significance of this scheme is to detect faster as it is a prime requirement of the real-time system to augment the strength of the network. The primary objective is to detect and to report it to the administrator to take corrective action to medicate the problem.

Cryptographic system alone is not enough to secure the network against the denial of service attack. Therefore, this Chapter contributes a knowledge based model building using RIPPER algorithm for detection of attacks aimed at network and computer to alleviate the bad behaviors of attackers in the network.

5.2 ISSUES AND CHALLENGES

One of the important issues in the communication network supporting real-time application is maintaining the same QOS throughout the
session. The denial of service attack aimed at the network by pumping lot of traffic to reduce the available bandwidth to other contenders is a very big challenge. The other types of DOS attack aimed at the system to render it useless by exploiting known bugs in the software like buffer overflow is also a very big challenge for real-time systems using shared network for communication. This becomes more challenging because of the underlying network protocol which was designed with no or minimal security consideration. Since, spoofing the IP addresses, other techniques such as war driving, differences in legal provisions of different countries and the border less nature of the network makes things very easy for the attacker to attack and escape without getting caught. The particular fact of able to attack the victim without physical contact gives many people the courage to attack the network and system.

Presence of sensors and detectors themselves discourage the attackers from attacking the system. Sophistications of technologies to block the unwanted traffic and other provisions to black list sites that are engaged in illegal activities demoralize attackers to keep away from systems and networks having necessary infrastructure to detect and deal with intrusion activities.

The construction of a classifier to detect various attacks normally requires lot of efforts and can be build only by experts. As new threats are found and reported every day, constructing new classifier manually is not cost effective. Reports often include exploited weakness in the system and other relevant facts and sequences leading to exploitation. In this situation it is better if an automated procedure is available to learn the classification rules from the pre-classified examples.
The challenge in providing solution to this must take care of the QOS requirements of the communication system supporting real-time application. That is, it must guarantee uninterrupted service throughout the procedure, provide minimum bandwidth guarantee, ensure delay within acceptable limits, maintain uniform delay or very minimum delay variation and acceptable packet loss ratio. Every subsystem involved in the communication infrastructure must be constructed to ensure these characteristics.

The intrusion detection models, which are conceived, created and studied, are mainly optimized for greater accuracy, producing less false positives and false negatives. Emphasize is not given for efficiency in terms of processing time. Aim of the proposed work is to create a classification model with RIPPER rule mining algorithm and to improve its time performance by implementing it for parallel processing in a multiprocessor system running symmetric multiprocessing (SMP) enabled operating system.

The proposed IDS model takes care of the above mentioned issues in the following ways:

- Automatically learns the classification rule from the pre-classified examples
- Ensures generation of less false positives
- Providing maximum classification accuracy
- Reduces the delay for intrusion detection by parallel implementation of the IDS
- Provides the smoothed delay variation by taking near uniform processing time for different types of attack
The objective of this proposed solution is to minimize the time taken for intrusion detection process and thereby ensuring minimum round trip delay in the communication. The other requirement of minimum delay variation is achieved independent of the type of attack.

5.3 FRAME WORK FOR CONSTRUCTION OF PARALLEL IDS

![Figure 5.1 Framework for construction of IDS using RIPPER](image)

The various components in the Parallel Integrated IDS framework is shown in Figure 5.1. Labeled attributes are the intrusion detection example data tagged with labels indicating whether it is attack or normal traffic. Rule learning algorithm is a module which constructs classification model from pre-labeled data set. Learned classification model is implemented separately as a sequential algorithm and parallel algorithm (Almasi and Gottlieb 1994). Parallel implementation is shown in Figure 5.1 as Parallel Integrated IDS. This module takes unlabelled attributes and classifies into various classes of traffic type.
A classification model is a sequence of rules usually ending in a default rule with an empty set of clauses. During classification, the left hand side of the rules are applied sequentially until one of them evaluates to true, and then the implied class label from the right hand side of the rule is offered as the class prediction.

5.3.1 Labeled Attributes

A labeled attribute consists of various features like time-based traffic features, host-based traffic features and content features about a connection together with label identifying the type of traffic. A connection is a sequence of TCP packets starting and ending at some well defined times, between which data flows to and from a source IP address to a target IP address under some well defined protocol. Each connection is labeled as either normal, or as an attack, with exactly one specific attack type. Each connection record consists of about 100 bytes.

Attacks fall into four main categories:

- **DOS**: denial-of-service, e.g. syn flood;
- **R2L**: unauthorized access from a remote machine, e.g. guessing password;
- **U2R**: unauthorized access to local superuser (root) privileges, e.g., various “buffer overflow” attacks;
- **probing**: surveillance and other probing, e.g., port scanning.

**Derived Features**

The “same host” features examine only the connections in the past two seconds that have the same destination host as the current connection, and calculate statistics related to protocol behavior, service, etc.
The similar “same service” features examine only the connections in the past two seconds that have the same service as the current connection.

“Same host” and “same service” features are together called time-based traffic features of the connection records.

Some probing attacks scan the hosts (or ports) using a much larger time interval than two seconds, for example once per minute. Therefore, connection records were also sorted by destination host, and features were constructed using a window of 100 connections to the same host instead of a time window. This yields a set of so-called host-based traffic features.

Unlike most of the DOS and probing attacks, there appear to be no sequential patterns that are frequent in records of R2L and U2R attacks. This is because the DOS and probing attacks involve many connections to some host(s) in a very short period of time, but the R2L and U2R attacks are embedded in the data portions of packets, and normally involve only a single connection.

Features constructed by examining suspicious behavior in the data portions, such as the number of failed login attempts by experts with domain knowledge are called “content” features.

5.3.2 Unlabeled Attributes

Unlabeled attributes are connection records with features like time-based traffic features, host-based traffic features and content features supplied as input to classifier to get the classification label identifying the type of traffic.
5.3.3 Ripper Rule Learning Algorithm

RIPPER was developed by William Cohen (1995) based on repeated application of Fumkranz and Widmer's (1994) Incremental Reduced Error Pruning (IREP) algorithm. IREP is a learning algorithm that used the basic concept of REP (Clifford Brunk and Michael Pazzani 1991, Giulia Pagallo and David Haussler 1990) with a modified separate-and-conquer rule learning algorithm and a new pruning technique. The Repeated Incremental Pruning to Produce Error Reduction (RIPPER) algorithm represents a significant performance improvement over previous rule induction algorithms. For a training set of size $n$, RIPPER's performance scales as $O(n \log_2 n)$ (William Cohen 1995).

In order for objects to be classified by most learning algorithms, they must first be transformed into a representation suitable for concept learning (Sam Scott 1998). All representations must consist of a vector of features, each describing some aspect of the objects to be classified.

In most machine learning systems, a feature may be either nominal (including binary) or continuous. Nominal features are those that take one of a finite number of pre-defined values, whereas continuous features are those that take on integer or real numeric values.

In RIPPER, a decision rule is defined as a sequence of Boolean clauses linked by logical AND operators that together imply membership in a particular class. The clauses are of the form $A = x$ or $A \neq x$ for nominal attributes and $A \leq y$ or $A \geq y$ for continuous attributes and $y$ is the value for $A$ that occurs in the training set.

A classification hypothesis is a sequence of rules usually ending in a default rule with an empty set of clauses. During classification, the left hand
side of the rules are applied sequentially until one of them evaluates to true, and then the implied class label from the right hand side of the rule is offered as the class prediction.

To explain the operation of RIPPER, the restricted case is considered in which the examples fall into one of two classes: positive or negative. A high level view of the algorithm is presented in the subsequent paragraphs.

To the basic algorithm, RIPPER adds several rule optimization steps as well as the option to improve the rule set by repeating the entire process for a number of iterations. Now an overview of the IREP algorithm is presented and additions will be discussed later.

**RIPPER Algorithm**

LOOP n TIMES
Start with the empty rule (TRUE => positive)
  LOOP UNTIL the stopping condition is reached
    Partition the training set into a growing set and a pruning set
    Grow a rule by greedily adding a clause to the left hand side guided by the grow heuristic
    Prune a rule by greedily deleting sequences of final clauses guided by the prune heuristic
    Remove examples covered by the rule from the training set
  END LOOP
Perform rule optimization on the entire rule set
END LOOP
The original IREP algorithm forms rules through a process of repeated growing and pruning. During the growing phase the rules are made more restrictive in order to fit the training data as close as possible. During the pruning phase, the rules are made less restrictive in order to avoid overfitting which can cause poor performance on unseen examples (Holte et al 1989).

IREP splits the training examples into a growing set and a pruning set (Cohen 1993). Rules to predict the positive class are grown one at a time by starting with an empty rule and then adding clauses to the left hand side in a greedy fashion under the guidance of a grow heuristic. Growing a single rule stops when it covers no negative examples from the growing set.

Each rule is pruned immediately after it is grown by deleting clauses that cover too many negative clauses in the pruning set under the guidance of a prune heuristic. After a new rule is grown and pruned, the covered examples are removed from both the growing and pruning set. Then the remaining data is repartitioned and another rule is grown. This rule growing process continues until all the examples in the training set are covered or some stopping condition is reached.

The bold-faced line in the RIPPER algorithm is new, while the rest of the algorithm is an implementation of IREP. During the rule growing phase, the goal is to add clauses greedily to an initially empty rule in such a way that the set of examples covered by the rule contains maximum positive examples and minimum negative examples.

The grow heuristic used in RIPPER is the information gain function proposed by Quinlan (1990). The algorithm starts with a set $T_0$ consisting of all examples remaining in the growing set. At the $i^{th}$ iteration of the rule
growing algorithm, the learner is working with a set $T_i$ consisting of $t_i^+$ positive examples and $t_i^-$ negative examples.

A measure of the information required to describe the class membership (equation 5.1) of all the examples is:

$$I(T_i) = -\log_2 \left[ \frac{t_i^+}{(t_i^+ - t_i^-)} \right]$$ (5.1)

The goal is to reduce the total amount of information. When a new clause $A_i$ is added to a rule, a new set of examples $T_{i+1}$ is formed consisting of all examples from $T_i$ covered by the new rule with $A_i$ added. Adding a clause can only restrict the coverage of a rule, therefore $T_{i+1}$ must always be a subset of $T_i$, though the new set may still contain both positive and negative examples. The information required to describe the new state is given in equation (5.2).

$$I(T_{i+1}) = -\log_2 \left[ \frac{t_{i+1}^+}{(t_{i+1}^+ - t_{i+1}^-)} \right]$$ (5.2)

If the addition of $A_i$ reduces the number of negative examples covered by the rule in relation to the number of positive examples, then the information required to describe the set will tend to decrease. A drop in the amount of information from $T_i$ to $T_{i+1}$ represents a gain in the amount of information contained in the rule under construction. But the goal is also to cover as many positive examples as possible, so the gain heuristic as given in equation (5.3) is defined as the number of remaining positive examples multiplied by the gain in information. So if $t_i^{++}$ of the positive examples from $T_i$ are still present in $T_{i+1}$ then:

$$\text{Gain}(A_i) = t_i^{++} \cdot (I(T_i) - I(T_{i+1}))$$ (5.3)
After a rule is grown, it will ideally cover many positive examples and no negative examples in the growing set. However, for noisy data it is not always desirable for the rules to fit the particular idiosyncrasies of the training data too closely. Rules that over fit the training data may perform poorly on unseen examples.

The function of the pruning stage, therefore, is to relax the rule so that it is more general and less prone to over fitting. In order to do this, the rule is tested against the examples in the pruning set. If the rule over fit the growing set, it will cover negative examples from the pruning set. The prune heuristic measures this coverage. The ideal condition is for the rule to cover many positive examples and no negative examples. Clauses are deleted to maximize the function shown in equation (5.4).

\[ v = \frac{p-n}{p+n} \]  
(5.4)

where \( p \) and \( n \) are the number of positive and negative examples in the pruning set that are covered by the rule.

Note that this value is maximized when \( n=0 \). In the original IREP implementations, clauses are deleted one by one in reverse order until no deletion can be found that increases the value of \( v \). RIPPER extends this process to consider dropping sequences of final clauses also.

Finally, a stopping condition is used to decide when to stop adding rules to the hypothesis. Furnkranz and Widmer propose a heuristic of stopping when the error rate of the next rule is greater than that of the empty rule. In RIPPER this heuristic is replaced with one based on the Minimum Description Length (MDL) principle from information theory (Quinlan 1995).
This criterion, described in (Ross Quinlan and Rivest 1989), is an elegant formula that balances accuracy against complexity based on the number of bits required to communicate complete and correct class information for a set of examples. The description length is obtained by adding the number of bits required to describe the classification hypothesis to the number of bits required to enumerate the exceptions to this hypothesis. Attempting to minimize this measurement will bias the learner towards simple and accurate rules. RIPPER stops adding rules when the new description length is more than 64 bits which is larger than the best description length so far.

As a refinement to the process described so far, RIPPER also includes two rule optimization steps. In the first step, each rule is considered in turn and two new potential replacement rules are grown. The first rule is grown and pruned starting with the empty rule as before and the second is grown starting with the original rule instead of the empty rule.

The main difference is that both rules are grown and pruned so as to optimize the error rate of the entire rule set on the entire pruning set. After the new rules are formed, the decision on, which one of the three candidate rules to include in the hypothesis is guided by the MDL heuristic. Finally, RIPPER simply repeats the entire algorithm a number of times to try to cover any remaining positive examples. The number of iterations of this process is subject to a parameter set by the user. In the current experiments the default value of 2 is always used.

The description of RIPPER given above is for a two-class problem. RIPPER handles multiple classes by ordering them from least to most prevalent and then treating each in order as a distinct two-class problem. So if the classes are ordered C1 to Ck, RIPPER first learns rules to distinguish the
least prevalent class C1 from classes C2 to Ck. Then all examples covered by these rules are removed, and RIPPER learns rules to distinguish C2 from C3 to Ck. This continues until only the most prevalent class Ck remains and this is used as the default class (William Cohen 1995). Note that this only applies to non-overlapping classes. When classes overlap, the only sensible choice is to train one binary classifier for each class, so the effect of RIPPER's class ordering means that rules will always be learned to cover the positive class first. The negative class will be left as the default.

The result obtained for the benchmark data with sequential and parallel implementation of classifier is tabulated as shown in Table 5.4. From the result as depicted in Figure 5.3 it is clear that parallel implementation of classifier saves considerable amount of time and provides smoothed delay variation which is the prime requirement of real-time communication system.

5.4 EXPERIMENT AND RESULT

As in the case of the previous experiment conducted for the evaluation of Parallel classifier using Decision Tree, this experiment too uses the same KDDCup1999 Intrusion detection contest data. For the completeness of the description, important aspects of the data set is described here. These data were prepared by the 1998 DARPA Intrusion Detection Evaluation program by MIT Lincoln Lab. Lincoln labs acquired nine weeks of raw TCP dump data. The raw data were processed into connection records, which consist of about 5 million connection records. The data set contain 24 attack types. These attacks fall into four main categories: Denial of service (DOS), Remote to user (R2L), User to root (U2R) and Probing.

The data set has 41 attributes for each connection record plus one class label. R2L and U2R attacks don’t have any sequential pattern like DOS
and Probe because the former attacks have the attacks embedded in the data packets whereas the later attacks have many connections in a short amount of time. Therefore, some features that look for suspicious behavior in the data packets like number of failed logins are constructed and these are called content features.

Experiments were conducted using Intel core 2 Due at 2GHz with 2GB main memory and 250GB hard disc drive in two phases, namely, a training and a testing phase. Training data set and testing data set are formed by splitting the available total data set into two sets. In the training phase RIPPER algorithm is used to construct a model using the training data of 370514 records of randomly selected 75 percent of total 494021 records to give maximum generalization accuracy. Model construction took 99325 seconds.

The test data were passed through the constructed model to detect the intrusion in the testing phase. Test was conducted by taking 100%, 75%, 50% and 25% of remaining data and studied the accuracy and time taken for classification for normal method and parallel method.

To gauge the time performance along with the accuracy, first 25% of test data is used. Classification accuracy obtained is 99.61, 33.33, 100, 99.77 and 99.99 percentages for corresponding PROBE, U2R, R2L, NORMAL and DOS as tabulated in Table 5.1. Test data set constituted of 257 records of PROBE, 3 records of U2R, 70 records of R2L, 24466 records of DOS and 6080 records of Normal traffic. Except the case of U2R, classification is more than 99 percentages. U2R consists of merely 3 records for testing and similar less number of cases for training and hence the generalization for unseen data is not possible. Processing time by the proposed parallel implementation achieved the reduction of about 1.68 times compared to sequential single processor implementation.
Table 5.1 Results with 25% of benchmark test data

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>PROBE</th>
<th>U2R</th>
<th>R2L</th>
<th>NORMAL</th>
<th>DOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misclassified cases</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>Total test cases</td>
<td>257</td>
<td>3</td>
<td>70</td>
<td>6080</td>
<td>24466</td>
</tr>
<tr>
<td>% of Accuracy</td>
<td>99.61</td>
<td>33.33</td>
<td>100</td>
<td>99.77</td>
<td>99.99</td>
</tr>
<tr>
<td>Time with two processors (ms)</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>122</td>
<td>473</td>
</tr>
<tr>
<td>Time with one processor (ms)</td>
<td>9</td>
<td>2</td>
<td>3</td>
<td>210</td>
<td>845</td>
</tr>
</tbody>
</table>

For the case of 50% of remaining data, classification accuracy exhibited is 98.44, 42.86, 100, 99.84 and 100 percentage for corresponding PROBE, U2R, R2L, NORMAL and DOS as tabulated in Table 5.2. Total test samples in each of DOS, PROBE, U2R, R2L and NORMAL are 48933, 514, 7, 141 and 12160. Number of misclassified records in each of these categories in the same order is 2, 8, 4, 0 and 19. The proposed parallel implementation took about 1.77 times less processing time compared to sequential single processor implementation.

Table 5.2 Results with 50% of benchmark test data

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>PROBE</th>
<th>U2R</th>
<th>R2L</th>
<th>NORMAL</th>
<th>DOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misclassified cases</td>
<td>8</td>
<td>4</td>
<td>0</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>Total test cases</td>
<td>514</td>
<td>7</td>
<td>141</td>
<td>12160</td>
<td>48933</td>
</tr>
<tr>
<td>% of Accuracy</td>
<td>98.44</td>
<td>42.86</td>
<td>100</td>
<td>99.84</td>
<td>100</td>
</tr>
<tr>
<td>Time with two processors (ms)</td>
<td>11</td>
<td>2</td>
<td>4</td>
<td>230</td>
<td>919</td>
</tr>
<tr>
<td>Time with one processor (ms)</td>
<td>16</td>
<td>2</td>
<td>4</td>
<td>416</td>
<td>1665</td>
</tr>
</tbody>
</table>
Classification accuracy for 75% of test data, which is 18.75 percent of actual data, is 99.98, 98.18, 60, 100 and 99.62 percentages for corresponding DOS, PROBE, U2R, R2L and NORMAL as tabulated in Table 5.3. Total number of records in each of DOS, PROBE, U2R, R2L and NORMAL are 73399, 771, 10, 212 and 18240. Number of misclassified records in each of these categories in the same order is 13, 14, 4, 0 and 70. Average performance gain obtained for parallel implementation is 1.77 times compared to sequential single processor implementation.

Table 5.3 Results with 75% of benchmark test data

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>PROBE</th>
<th>U2R</th>
<th>R2L</th>
<th>NORMAL</th>
<th>DOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misclassified cases</td>
<td>14</td>
<td>4</td>
<td>0</td>
<td>70</td>
<td>13</td>
</tr>
<tr>
<td>Total test cases</td>
<td>771</td>
<td>10</td>
<td>212</td>
<td>18240</td>
<td>73399</td>
</tr>
<tr>
<td>% of Accuracy</td>
<td>98.18</td>
<td>60.00</td>
<td>100</td>
<td>99.62</td>
<td>99.98</td>
</tr>
<tr>
<td>Time with two processors (ms)</td>
<td>15</td>
<td>2</td>
<td>5</td>
<td>341</td>
<td>1368</td>
</tr>
<tr>
<td>Time with one processor (ms)</td>
<td>22</td>
<td>2</td>
<td>5</td>
<td>617</td>
<td>2478</td>
</tr>
</tbody>
</table>

When 100% of remaining data, that is 25 percent of actual data were tested, obtained accuracy is 99.91%, 98.63%, 42.85% and 99.29% for corresponding DOS, PROBE, U2R, R2L and NORMAL as tabulated in Table 5.4. Total number of records in each of DOS, PROBE, U2R, R2L and NORMAL are 97866, 1028, 14, 288 and 24321. Number of misclassified records in each of these categories in the same order is 86, 14, 8, 2 and 74. By perusing the accuracy, for U2R we are getting less accuracy. But this occurs because number of records available for training and as well as testing is very meager. This justifies the deficiency in the obtained rules for this class. We repeated the experiment with the parallel model, we obtained the improvement of 1.75 in the time taken to classify all these test cases compared to the sequential single processor run.
Table 5.4 Results with 100% of benchmark test data

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>PROBE</th>
<th>U2R</th>
<th>R2L</th>
<th>NORMAL</th>
<th>DOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misclassified cases</td>
<td>14</td>
<td>8</td>
<td>2</td>
<td>74</td>
<td>86</td>
</tr>
<tr>
<td>Total test cases</td>
<td>1028</td>
<td>14</td>
<td>283</td>
<td>24321</td>
<td>97866</td>
</tr>
<tr>
<td>% of Accuracy</td>
<td>98.64</td>
<td>42.86</td>
<td>99.29</td>
<td>99.70</td>
<td>99.91</td>
</tr>
<tr>
<td>Time with two processors (ms)</td>
<td>22</td>
<td>2</td>
<td>6</td>
<td>461</td>
<td>1853</td>
</tr>
<tr>
<td>Time with one processor (ms)</td>
<td>32</td>
<td>2</td>
<td>6</td>
<td>823</td>
<td>3395</td>
</tr>
</tbody>
</table>

In any environment, DOS attack is very widely noticed and causes so much loss to organizations world over in terms of money and reputation. As this menace can be launched by any novice without much technological skill with ready made tools available in the Internet and can cause severe performance degradation, every organization must be armed with fighting this efficiently. The number of test cases available numbering 97866 is also reflective of actual scenario. The results as evident from the Figure 5.2 are very encouraging with remarkable reduction in time for the parallel implementation compared with sequential implementation.

![DOS Attack Graph](image)

Figure 5.2 Performance for detection of DOS attack
Total number of test samples in the normal traffic category is 24321. Performance gain progresses along with the increase in considered test samples as clearly visible from Figure 5.3. This category of traffic should be more in an ideal environment and hence the reduction in processing time achieved for the proposed method is significant justification for the network supporting real-time traffic which is to provide service with stringent timing guarantees. The time reduction of 88 ms realized for 6080 records by the proposed parallel implementation compared to the sequential single processor implementation execution time of 210 ms is significant considering the application. The result obtained is still encouraging when number of test sample increases as observable from the Figure 5.3. For the test samples of size 24321, the time reduction achieved for the proposed parallel implementation is 362 ms when sequential single processor implementation consumes execution time of 823 ms.

![NORMAL](image)

**Figure 5.3 Performance for detection of normal traffic**

Number of test samples varies from 70 to 283 in the R2L attack category. This number is not a real candidate to show any appreciable performance improvement considering the initial setup time required to start
the process. Even then small performance gain observable is a good justification for the selection of the proposed method as seen from the Figure 5.4.

![R2L Attack](image)

**Figure 5.4 Performance for detection of R2L attack**

Available test cases in the U2R attack range from 3 to 14 as plotted in Figure 5.5. The result are in line with the expectation for this small number of test cases as initial setup cost out weigh the actual processing of classifier code. So, no reduction is observable in the experiment.

![U2R Attack](image)

**Figure 5.5 Performance for detection of U2R attack**
Reduction in time to detect PROBE attack is depicted in the Figure 5.6. PROBE is normally a first attack in any network. Since, when the attacker plan to attack a system or network, first task would be to gather information about various aspects of the network and systems like available systems, network topology, operating systems(OS) running on different machines, version number and release of OS, applications running on different systems, open ports, etc. In any network this attack may be expected to occur frequently. Many networks or system administrators may not take it serious to take any active steps immediately because it causes no serious problems to the normal operation. Administrator may note down the activity for further monitoring of any suspicious activity from that source. So, it is important to detect and notify to the administrators to prepare for future monitoring of these sources for the forensic analysis and to gather evidence to prove there was malicious intention from that source. When data records got increased observed time reduction for parallel implementation compared to the sequential single processor implementation shows sustained performance.

![Figure 5.6 Performance for detection of probe attack](image)
5.5 SUMMARY

Issues and challenges related to communication system supporting real-time application are discussed in this chapter. The framework for the proposed intrusion detection system is depicted. Constituents of the framework are elaborately discussed. The supervised learning method for automatic construction of classifier from the pre-classified attribute, RIPPER algorithm is explained. Attributes used in the study and its selection and preparation are explained. Experimental method is illustrated and results are discussed.

As the result shows, the proposed method gives expected speed up to the intrusion detection process at the desired level. Results also show DOS and PROBE attacks are detected more than 99 percent. Other attacks look as if exhibiting poor performance but in reality considering the number of cases available for training and testing indicates this is not a poor performance. As also considering the application, the computer used in these systems will not be providing any other service other than the specific service meant for the real-time system. So, perceived threat of U2R and R2L are small. More over, these specialized applications will be available only with the institutions offering such a service and hence its exposure to large number of people for vulnerability analysis will be very limited. Considering all these factors, the proposed system will serve the intended purpose at the desired time limit.