A CONTENT-ADDRESSABLE-MEMORY ASSISTED INTRUSION PREVENTION EXPERT SYSTEM FOR GIGABIT NETWORKS

by

Ying Yu

B.S., Fudan University, 1997

M.S., Fudan University, 2000

Submitted to the Graduate Faculty of

School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2006

UNIVERSITY OF PITTSBURGH

SCHOOL OF ENGINEERING

This dissertation was presented

by

Ying Yu

It was defended on

August 25, 2006

and approved by

Dr. Raymond R. Hoare, Assistant Professor, Department of Electrical & Computer Engineering

Dr. Steven P. Levitan, Professor, Department of Electrical & Computer Engineering

Dr. James T. Cain, Professor, Department of Electrical & Computer Engineering

Dr. Ronald G. Hoelzeman, Professor, Department of Electrical & Computer Engineering

Dr. Brady Hunsaker, Department of Industrial Engineering

Dissertation Director: Dr. Alex K. Jones, Assistant Professor, Department of Electrical & Computer Engineering

A CONTENT-ADDRESSABLE-MEMORY ASSISTED INTRUSION PREVENTION EXPERT SYSTEM FOR GIGABIT NETWORKS

Ying Yu, Ph.D.

University of Pittsburgh, 2006

Cyber intrusions have become a serious problem with growing frequency and complexity. Current Intrusion Detection/Prevention Systems (IDS/IPS) are deficient in speed and/or accuracy. Expert systems are one functionally effective IDS/IPS method. However, they are in general computationally intensive and too slow for real time requirements. This poor performance prohibits expert system's applications in gigabit networks.

This dissertation describes a novel intrusion prevention expert system architecture that utilizes the parallel search capability of Content Addressable Memory (CAM) to perform intrusion detection at gigabit/second wire speed. A CAM is a parallel search memory that compares all of its entries against input data in parallel. This parallel search is much faster than the serial search operation in Random Access Memory (RAM). The major contribution of this thesis is to accelerate the expert system's performance bottleneck "match" processes using the parallel search power of a CAM, thereby enabling the expert systems for wire speed network IDS/IPS applications.

To map an expert system's Match process into a CAM, this research introduces a novel "Contextual Rule" (C-Rule) method that fundamentally changes expert systems' computational structures without changing its functionality for the IDS/IPS problem domain. This "Contextual Rule" method combines expert system rules and current network states into a new type of dynamic rule that exists only under specific network state conditions. This method converts the conventional two-database match process into a one-database search process. Therefore it enables the core functionality of the expert system to be mapped into a CAM and take advantage of its search parallelism.

This thesis also introduces the CAM-Assisted Intrusion Prevention Expert System (CAIPES) architecture and shows how it can support the vast majority of the rules in the 1999 Lincoln Lab's DARPA Intrusion Detection Evaluation data set, and rules in the open source IDS "Snort". Supported rules are able to detect single-packet attacks, abusive traffic and packet flooding attacks, sequences of packets attacks, and flooding of sequences attacks.

Prototyping and simulation have been performed to demonstrate the detection capability of these four types of attacks. Hardware simulation of an existing CAM shows that the CAIPES architecture enables gigabit/s IDS/IPS.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

This work would not have been possible without the support and encouragement of my Ph.D. advisor, Dr. Raymond R. Hoare. For more than 5 years, he has always been there to actively support this work with ideas, resources, and direction. I am very grateful for his patient supervision and great inspiration. Dr. Alex K. Jones, my dissertation advisor in the final stages of the work and also chair of the committee, has also been abundantly helpful. I want to thank him especially for his insightful advice and technical discussions that facilitated this work.

I would also like to express my gratitude to my committee members: Dr. Steven P. Levitan, Dr. James T. Cain, Dr. Ronald G. Hoelzeman, and Dr. Brady Hunsaker, thank you all for taking time to offer guidance and support for this work.

I am also grateful to many other colleagues and staff in the department who helped me in many ways during my research.

Finally, I would like to thank my parents for their constant encouragement and love that I have relied on throughout the long journey of Ph.D. study.

1.0 INTRODUCTION

1.1 NETWORK INTRUSIONS AND NETWORK SECURITY

Computer and network intrusions can be defined as "any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource" [1]. Cyber intrusion has become a serious problem with its growing frequency and complexity [2-4]. For example, the number of computer intrusions is almost doubling every year [5, 6]. 90% of corporations and government agencies detected security incidents in 2002 [7], rising from 70% in 2000 [5] and 42% in 1996 [5]. Intrusions also cause large amounts of financial loss: in 2003, virus damage resulted in 55 billion dollars in loss [8]. In addition, the threat of intrusion has increased due to the availability of more hacking tools, which reduce the technical skills required to launch an attack during the last 10 years while the sophistication of those attacks has risen

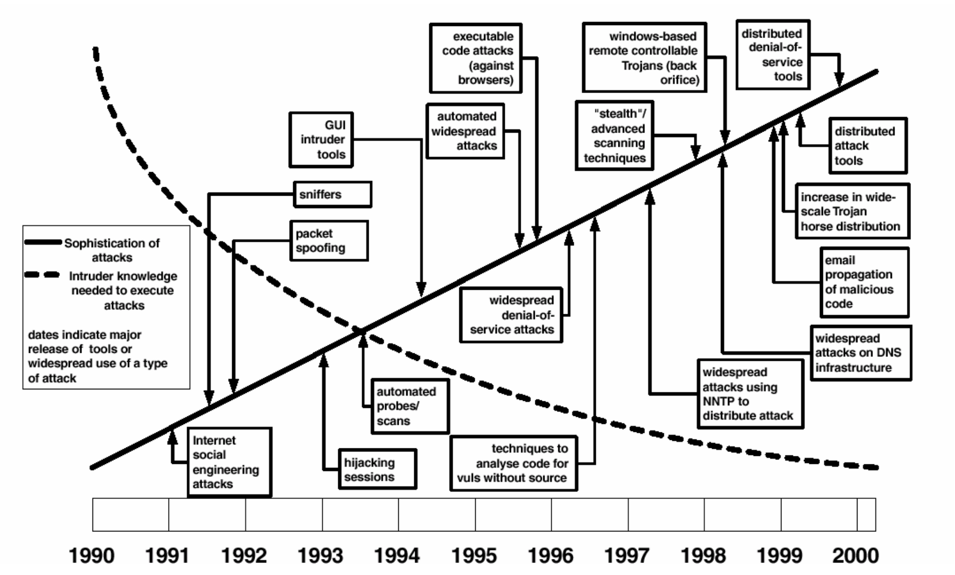

executable code attacks (against browsers)
windows-based remote controllable Trojans (back orifice)
distributed denial-of-service tools
GUI intruder tools
automated widespread attacks
"stealth"/ advanced scanning techniques
distributed attack tools
sniffers
increase in wide-scale Trojan horse distribution
Sophistication of attacks
Intruder knowledge needed to execute attacks
dates indicate major release of tools or widespread use of a type of attack
packet spoofing
email propagation of malicious code
widespread denial-of-service attacks
widespread attacks on DNS infrastructure
widespread attacks using NNTP to distribute attack
automated probes/ scans
Internet social engineering attacks
hijacking sessions
techniques to analyse code for vuls without source
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000

Figure 1.1 The Evolution of Attack Sophistication and Devolution of Attack Skills [11]

over the same time, as shown in Figure 1.1 [5, 11]. This trend is expected to continue. All these facts lead to a need for better network security solutions [5].

A CSI/FBI security report states that 90% of attacks bypass firewalls [11]. Therefore, *intrusion detection*, a mechanism to discover violations of systems' security policies, with its outgrowth *intrusion prevention*, a technique to actively prevent intrusions, has been proposed since 1980 as a necessary complement to current prevention-based security solutions [5, 6]. Intrusion Detection Systems (IDS) and later Intrusion Prevention Systems (IPS) have created rapidly growing markets as well as dynamic research areas [3, 6, 19]. However, current IDS and IPS remain deficient in speed and/or accuracy [7, 29, 71, 72].

1.2 INTRUSION DETECTION HISTORY AND FOUNDATION WORK

Intrusion Detection Systems (IDS) has become a rapidly growing market as well as a dynamic research area [3, 6, 19] since its birth in 1980 [5, 6]. Before that, manual monitoring and analysis of numerous audit files was required to detect suspicious activities [15]. Important milestones in intrusion detection histories are introduced as follows:

In 1980, Anderson [20] proposed recording some specific audit trail information representing the computer security status for statistical analysis. This is now regarded as the beginning of intrusion detection technology [6]. In 1987, Denning [21] first proposed an Intrusion Detection model as a solution to computer security problems [5]. The "*Intrusion Detection Expert System (IDES)*" model uses a hybrid expert system and statistical approach to analyze audit trails on a host [21]. It laid the groundwork for future research and commercial IDS [6]. In 1990, Heberlein *et al.* [22-24] first introduced the concept of "*Network Intrusion Detection* (NID)" and "*network-based*" IDS, which passively monitors network traffic on LAN instead of inspecting audit trails on a host computer [22-24] as the conventional *host-based intrusion detection* does. Most commercial IDS today use directly observed network data as their main (or only) data source [11]. In 1991, Snapp *et al.* [25] first proposed the concept of a "*Distributed Intrusion Detection System (DIDS)*", which includes host-based IDS on each host, network-based IDS on each LAN, and a central manager to find suspicious behavior among all information from sensors. The use of IDS in distributed environments has attracted much research interest since then. In 1995, Crosbie *et al.* [26] proposed applying autonomous agents to further improve the scalability, maintainability, efficiency and fault tolerance of IDS. A large number of novel techniques have been researched in recent years, focusing on novel various detection techniques.

When considering detection techniques, IDS can be basically classified into two categories [5, 11, 15, 27]: *misuse detection* and *anomaly detection*. *Misuse detection* specifies bad or unacceptable activities by applying knowledge of known intrusions [11, 15, 27] to recognize similar intrusive activities contained within inspected data. The main advantage of misuse detection is its comparatively accurate detection with low false positive rate, i.e. false alarms on non-intrusive behavior [14, 15]. The main limitation is that it can only identify known attacks whose signatures are stored in the database [14, 15]. New signatures need to be invented and added to the knowledgebase after novel attacks first appear. *Anomaly detection* defines normal traffic pattern and normal behavior based on historical activities over a period of time [11, 13-15, 19, 27] and then detects any deviation, which will be interpreted as intrusive behavior [13, 14, 19]. The main strength of anomaly detection is that it can detect new and unknown attacks by identifying them as different from "normal behavior" [13, 14, 19]. The limitations of anomaly detection include its high false positive rate [13, 14, 19] and potential chances for an attacker to gradually train the IDS to regard the intrusive behavior as normal [14].

Many research and commercial IDS products have been developed using either misuse detection or anomaly detection, or both.

1.3 EXPERT SYSTEMS IN IDS

Expert systems have been used in IDS ever since Denning's IDES model in 1987 [21]. Since an expert system contains a human's expertise knowledge about attacks, it can identify malicious behavior in its monitoring data. Employing a range of internal representations of attack knowledge, expert systems have been used in more than ten misuse detection systems, several anomaly detection systems, and even some novel state-transition detection systems. Most of these are host-based detection systems. Details of expert system applications in IDS can be found in Chapter 2.

The expert systems approach differs from simple pattern matching [43, 47], which is a popular method currently used in an open source light weight IDS called Snort [48]. Though both expert systems and simple pattern matching describe characteristics of intrusive behavior and pattern match current events against these description signatures [43], their reasoning strategies and knowledge representations are quite different [43].

Simple pattern matching uses stateless reasoning and single-event-based attack signature representation [43, 47]. This enables its effective detection ability for single-step intrusions occurring within one event, with fast speed and low system memory requirement. But it is very inaccurate in detecting complicated multi-step intrusions because of the inability of its rules to describe these intrusions accurately without also covering any non-intrusive activities [43, 47]. Because of its limited signature expressablity, it is typically only used in stateless rule-based misuse detection, and used to detect single-packet-attack in networks.

Expert systems use stateful reasoning and a multi-event-based attack signature representation [43, 46, 47]. Expert systems are excellent for searching among multiple events for a specific sequence of events, or single events exceeding a normal threshold [43, 47]. Therefore, they have the ability to express complicated multi-step or distributed intrusive behavior, which simple pattern matching does not [43, 47]. However, due to its computational intensive nature and statefulness, expert systems are usually much slower than pattern matching, and require more system memory to store system states [43, 46]. However, the rich expressablity enables expert systems to be used in different detection methods, and to detect multi-packet or distributed intrusions in networks.

Table 1.1 Comparison of Pattern Matching Detection and Expert Systems Detection

	Pattern Matching	**Expert Systems**
Reasoning [43,46,47]	Stateless	Stateful
Attack coverage [43,46,47]	Single-packet attack	Single or multi-packet attack scenario, anomalies
State Management [43,47]	No need or less burden to system	Heavy burden to system
Rule Feature [43,47]	Simple and Short	Sophisticated
Expressability [43,47]	Limited	Rich
Rule Coverage [43,47]	Few intrusion variations	Many intrusion variations
Rule Set [43,47]	Inflated	Concise
Detection Speed [43,46,47]	Faster	Slower

Since intrusions are gaining in complexity as shown in Figure 1.1, simple pattern matching cannot achieve satisfactory detection results. Expert systems have better detection ability because of their stateful reasoning and rich expressability. In addition, heterogeneous detection methods can be easily integrated into one expert system by having additional rules. This makes the expert system approach more promising in the long run.

1.4 PROBLEM STATEMENT

Expert systems are one existing IDS method that is functionally effective in detecting network intrusions [16, 45]. They have been applied to more than ten IDS since Denning's IDES model in 1987 [21, 24, 45]. Since it contains a human expert's knowledge of attacks, an expert system can identify malicious behavior in the data it monitors. With different representations of attack knowledge, expert systems have been used in more than ten misuse detection systems (i.e., IDS that use "bad" behavior signatures to find matched intrusive behavior), and several anomaly detection systems (i.e., IDS that use "good" behavior profiles to find deviant behavior).

However, expert systems in general are computationally intensive and usually too slow for high speed or real-time demands [51, 60]. Existing expert systems require 10^{-3} to 10^{-6} seconds to process one input datum. This performance is insufficient to keep up with gigabit network traffic where only 10^{-7} seconds can be spent processing each packet. Therefore, expert systems' low performance prohibits their applications in wire-speed IDS/IPS in gigabit networks [45, 46]. For example, based on a P-Best expert system, host-based IDS (i.e., IDS using host audit trails as monitoring sources) can achieve only 12Mbps throughput with 2 rules, and 6Mbps with 28 rules [45]. Network-based IDS (i.e., IDS using network traffic as monitoring sources) can achieve only 30Mbps throughput with 12 rules [45]. In addition, performance will further diminish when more attack signatures are added.

Therefore, the performance problem in expert systems has limited their applications in wire speed network Intrusion Detection and Prevention Systems. This thesis shows how the parallel search ability of a CAM can be used to accelerate IDS/IPS expert systems. The following five issues were studied for this design. First, how can the map range matching criteria in network IPS rules be mapped into a single-cycle CAM operation? Second, based on the solution of range matching in a CAM, how can the basic network IPS functions of single-packet attack detection be mapped into a CAM? Third, based on the single-packet detection solution, how can complicated network IPS functions of multiple-packet attack detection be mapped into a CAM? Fourth, when solving the multiple-packet detection issue, can we create a general architecture for network IPS expert system that can be mapped into CAM? Finally, how can we simulate and prototype system to demonstrate the CAIPES functionality?

1.5 THESIS STATEMENT

It is possible to accelerate computationally intensive expert systems using a highly parallel search memory called Content Addressable Memory (CAM), thereby enabling expert system-based network IDS/IPS to detect intrusions at gigabit/second wire speed.

In order to apply the CAM in network IPS expert systems implementation, each of the five issues identified in the preceding section were addressed. First, to map range matching criteria in network IPS rules, we invented a new encoding scheme so that an arbitrary range can be matched in a single cycle using CAM. Second, based on the range matching solution, we mapped basic network IPS functions of single-packet attack detection into a CAM. Third, based on the single-packet matching solution, we mapped complicated network IPS functions of multiple-packet attack detection into CAM. Fourth, when solving the multiple-packet detection, we designed a general architecture of a "contextual rule" method for network IPS expert system to be mapped into CAM, and created the CAM-Assisted Intrusion Prevention Expert System (CAIPES). Lastly, we built a prototype system to demonstrate the CAIPES functionality and performance. We have chosen Lincoln Lab intrusion detection evaluation dataset [69] for the simulation.

1.6 THE PROPOSED EXPERT SYSTEM PLATFORM FOR NETWORK INTRUSION PREVENTION SYSTEM

A function diagram of the CAIPES expert system platform described in this thesis is shown in the light gray box of Figure 1.2. The expert system will monitor each network packet in the traffic flow from a protocol analyzer where the protocol information is extracted from the packet. The expert system makes decisions on the current packet and labels it with actions that should be taken on the packet. For example, good packets are labeled as "pass", bad packets are labeled as "drop", and suspicious packets are labeled as "low priority" and will be sent to a low priority queue.

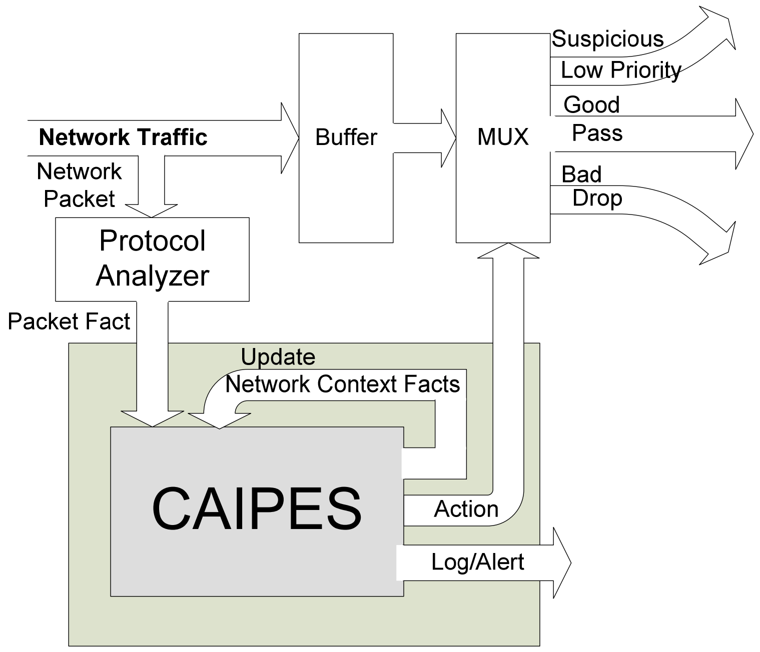


Figure 1.2 Function Diagram of the Expert System Platform for NIPS

1.7 CONTRIBUTIONS AND ORGANIZATION OF THE DISSERTATION

In this dissertation, we describe the CAM-Assisted Intrusion Prevention Expert System (CAIPES) that has detection capability for both single and multi-packet attacks. The structure of the dissertation, as well as the major contributions of this research is described below:

1.7.1 A Novel Encoding Scheme for Range Matching Using Content Addressable Memory

The parallel search engine CAM has the ability to perform exact matching in parallel with a high performance. However, Intrusion Detection and Prevention usually requires extra range matching functions besides exact matching. Single-cycle range matching of binary code cannot be effectively mapped into a CAM because it requires a large number of CAM entries. We introduce a new encoding scheme by mixing the features of one-hot coding and Johnson coding that enables single-cycle range matching in a Ternary CAM with a maximum of three CAM entries required for any arbitrary range. This novel encoding scheme is more efficient for single-cycle CAM range matching in terms of CAM width and CAM depth required. The details of this hybrid encoding scheme are covered in Chapter 3.

The novel encoding scheme solves the issue of how to map the range matching criteria of IPS rules into a CAM. The scheme is the foundation of the following single-packet and multi-packet detection functions. This novel encoding scheme will be applied to map the detection rules into CAM when range matching criteria are involved in expert system detection rules.

1.7.2 Detection of Single-Packet Attack Using CAM

This research designed an IPS expert system architecture that utilizes a highly parallel search memory CAM to achieve detection rate for gigabit networks at wire speed. Detection of single-packet attack is mapped to a CAM, including detection ability for network packet header checking and payload string matching. This architecture supports up to 91% of Snort rules (version 2.0) [82], where 17.9%-28.4% of the rules are solved by the range mapping scheme mentioned above. The details of the single-packet attack detection are covered in Chapter 4.

Single-packet attack detection is the foundation of the detection of multi-packet attack, where not only the single-packet-based rules are mapped to the CAM, but also previous packet information is used in the detection and are also mapped into CAM.

1.7.3 Detection of Multi-Packet Attack in the CAIPES Architecture

Chapter 5 introduces a new classification for network intrusions based on a detection system's perspective, and classify the multi-packet attacks from the Lincoln Lab IDS evaluation dataset into the following three categories: flooding of packet attacks, sequence of packets attack, and flooding of sequence attack. The CAIPES architecture supports detection methods toward these three multi-packet attacks. Expert system detection rules here include not only current packet information but also previous packet information, and are mapped into CAM. The details of multi-packet attack detection in CAIPES are covered in Chapter 5.

1.7.4 A Novel Contextual Rule Method to Apply CAM to Expert Systems

This research introduces a novel dynamic "Contextual Rule" method for implementing multi-packet detection in CAM. This method fundamentally changes expert systems' computational structures but remains functionally identical for the IDS/IPS problem domain. It converts the expert systems' match process between two-databases into a single database match process according to the features of the IDS/IPS problem domain. Thus, the parallel searching capability of a CAM can be applied to the one-database search execution in expert systems. This method enables utilization of a CAM's parallel search capability to accelerate the computationally intensive *match* process in the expert systems. The details of the Contextual Rule method and its design issues are covered in Chapter 5.

This novel Contextual Rule Method provides a general architecture for expert systems in network-based IPS problem domain to use CAM. Therefore, the multi-packet detection towards three attack categories can be solved in CAM using this unified architecture.

1.7.5 Demonstration of Extensibility and Proof-of-Concept Prototype

The CAIPES platform is extensible for new rules and new detection methods without any change in hardware. In this thesis, a prototype has been developed in the System C language. Nine detection methods chosen from the three multi-packet attacks categories have been implemented in the prototype system. We use traffic sniffing data from Lincoln Lab IDS evaluation dataset as the simulation data. A total of 4 MB of attack traffic data and 7 MB of clean traffic data have been run through the simulation. The simulation results show that CAIPES has performance to keep up with Gigabit detection. The details of simulation for the three attack categories are also covered in Chapter 5.

Conclusions and future directions are described in Chapter 6.

2.0 LITERATURE REVIEW

2.1 ENABLING TECHNOLOGY: CONTENT ADDRESSABLE MEMORY

Content Addressable Memory (CAM) [77] is a dedicated, highly parallel search memory that can search thousands of times faster than a CPU or Random Access Memory (RAM). In RAM, finding a particular data word takes many cycles, as a RAM performs serial searches among all entries and only fetches and compares one entry at a time. This process is shown in Figure 2.1a.

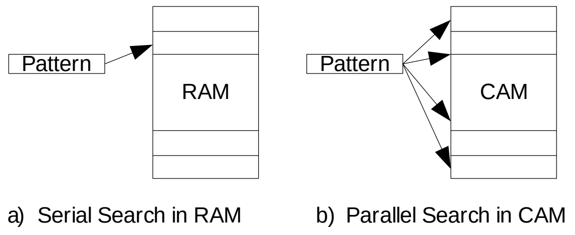


Figure 2.1 Random Access Memory (RAM) and Content Addressable Memory (CAM)

On the other hand, finding a particular word in CAM takes only a single cycle. A CAM performs parallel searches among all entries, and checks a given pattern against thousands of memory entries at the same time, as shown in Figure 2.1b. After the parallel search, a CAM produces memory addresses whose content matches the input pattern. As an example of a CAM's performance, an IDT Network Search Engine IDT75K62100 CAM [85] can search a 144b data pattern 100 million times per second on a 9Mb database, thus achieving an input processing speed of 14.4Gbps.

A Ternary CAM can store three kinds of values: binary '1', binary '0', and "don't care" for each memory bit. The "don't care" bit can be used as a mask for CAM comparisons; a stored "don't-care" bit would always result in a match with any input pattern bit.

In this thesis, "CAM" is used to indicate commercially available Ternary CAMs. To be accurate, let us define the functions that a "CAM" usually has and those which it normally lacks as follows:

A CAM usually can perform the following functions:

- Single match to find the first data entry in it that matches the input data
- Multi-match to find all data entries that match the input data, with matched results extracted one by one, serially by scroll operations.
- Writing to one CAM entry, just as a RAM write function
- Invalidating all the entries that match the input data in parallel
- Learning a new data entry if this data is not in the CAM

A CAM typically doesn't have functions of:

- Counting the number of matched results in a Multi-Match operation
- Bit vectors to show Multi-match results in parallel in single cycle
- Bit serial comparison when matching
- Parallel writing

A commercially available CAM IDT network search engine IDT75K62100 CAM [85] is used in this research. It has a total of 9Mbits with each entry configurable as 72 bits wide, 144 bits wide, 288 bits wide or the longest available width of 576 bits. Its internal clock is 10ns wide and 100Mhz. Single search operation can be issued in pipeline. It can achieve an input processing speed of 14.4Gbps.

2.2 EXPERT SYSTEMS

An expert system is defined as "a computer program that represents and reasons with knowledge of some specialist with a view to solving problems or giving advice" [41]. Being a branch of Artificial Intelligence, expert systems were first developed in the 1970s and have been widely applied to business, medicine, engineering, and science in order to solve complicated problems intelligently [42].

The essence of a network IPS expert system is shown in Figure 2.2. Network traffic is sent to the expert system, packet by packet. The expert system analyzes the traffic and decides whether it's good, bad, or suspicious. The expert system then takes actions to pass, drop, or send packets to a low priority queue respectively. The expert system acquires expert knowledge from human experts and usually represents it in rule format. This knowledge is stored in a rule base (also called a Knowledge Base). The fact base (also called Working Memory), is a global database that stores the facts on which the system bases decisions. The inference engine reasons and makes decisions according to existing facts and rules.

In IDS/IPS applications, the inference engine usually uses a “forward-chaining reasoning” that uses facts to draw a conclusion. The inference engine executes according to a “Match-Select-Act” cycle, as shown in Figure 2.2. During the “Match” phase, the engine searches the fact base and rule bases to find rules that are satisfied by the existing facts. In the “Select” phase it selects one rule among all the matched rules, usually the highest priority rule. In the “Act” phase it executes the selected rule’s action. This completes one execution cycle of the expert system. Processing an input data requires one or more Match-Select-Act cycles, until all matched rules are executed.

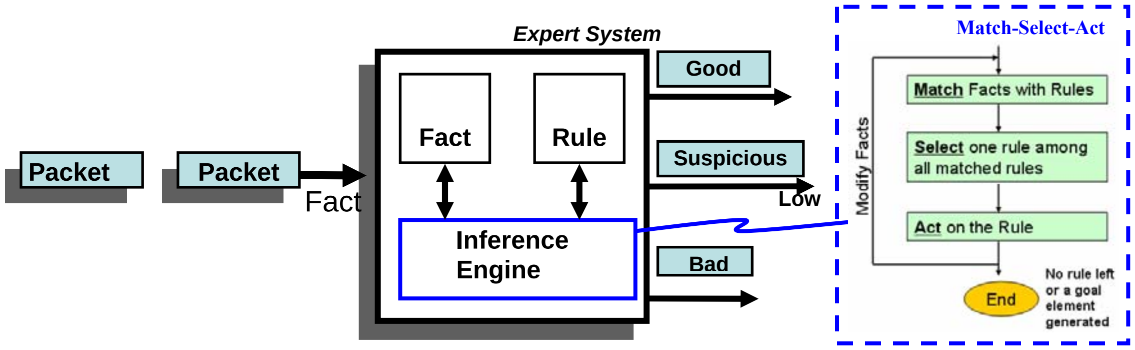


Figure 2.2 Basic Elements of Network IPS Expert System and Its Execution

An expert system rule has two parts: the conditions and the actions. The conditions directly follow the “if” in the statement of a rule. The conditions must be satisfied in order for a rule to be fired. Actions directly follow the “then” in a rule’s statement and indicate what to do when the rule’s conditions are met.

To detect step 1 in the 3-way handshake communication in a TCP connection establishment

IF *Current_Packet (Protocol ==TCP) and (Flag== SYN)* **--Conditions**
THEN *Add TCP_State (State = 1)* **--Actions**

Figure 2.3 An Example of an Expert System Rule

Figure 2.3 shows a simplified example of an expert system rule designed to detect the 1st step of a 3-way handshake communication in TCP connection establishment. In this example, the expert system checks the incoming packet. If the packet satisfies all the conditions: the "protocol" field of the packet is TCP and the "flag" field of the packet has only SYN flag set, then one action can be executed: i.e. add a fact "TCP_state" with the field "state" equal to 1 to the fact base of the expert system. This action records the occurrence of the 1st step of TCP connection in the expert system.

2.3 MATCH ALGORITHMS

In the Match-Select-Act execution cycle shown in Figure 2.2, the match phase is the most computationally intensive and creates a performance bottleneck that takes 50%-90% of the total execution time [50-53]. The match process in expert systems performs searches between two databases: the fact base and rule base, each of which usually contains hundreds of items [42]. The match process involves searches for any combinations of facts that satisfy any rules (shown in Figure 2.4).

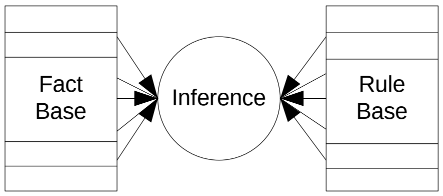


Figure 2.4 Matching Problem in Expert Systems

There are three existing match methods, shown in Figures 2.5, 2.6, 2.7. Details of each method are explained in the following sections.

2.3.1 Brute Force Algorithm with Rules Searching the Fact Base

Description: Figure 2.5 shows a brute-force matching method with rules searching the fact base. It is a software-based solution. In this algorithm, each rule searches the entire fact base to determine if the condition of the rule is satisfied by any combinations of facts in the fact base.

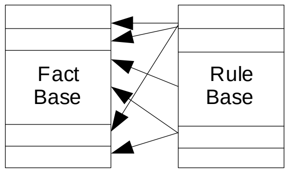


Figure 2.5 A Brute Force Algorithm with Each Rule Searching the Fact Base [54, 60]

Example: Figure 2.6 shows two rules to detect the 1st and 2nd steps in a 3-way handshake communication of a TCP connection establishment. These two rules will be used as an example of expert systems' match process in all 3 existing match methods.

```
Detection of step 1 and 2 in the 3-way handshake communication in a TCP connection establishment

Rule 1:  to detect step 1
IF  Current_Packet (Protocol ==TCP) and (Flag== SYN)                          --Condition
THEN Add TCP_State (State = 1)(Client=Current_Packet.SIP&Port)                --Actions
                   (Host= Current_Packet.DIP&Port)(Seq<= Current_Packet.Seq)

Rule 2:  to detect step 2
 IF  Current_Packet (Protocol ==TCP) and (Flag==SYN_ACK)                      --Condition 1
          and TCP_State (State ==1)
     and (TCP_State.Client == Current_Packet.SIP&Port)                        --Condition 2
     and (TCP_State.Host == Current_Packet.DIP&Port)                          --Condition 3
     and (TCP_state.  Seq = =Current_Packet.Ack-1);                           --Condition 4
THEN  modify TCP_State (State =2)(Seq = Current_Packet.Seq)                   --Actions
```

Figure 2.6 Two Rules to Detect Step 1 and 2 in the 3-Way Handshake Communication in a TCP Connection Establishment

Suppose there are 400 facts in the fact base. To match the first rule, each fact in the fact base needs to be fetched and tested to see if it satisfies the condition in Rule 1. This leads to 400 RAM accesses.

To match the second rule, condition 1 requires 400 RAM accesses to find a matched "current_packet" fact. Conditions 2 and 3 are more complicated. Not only should they check whether a TCP_State fact's field "state" is equal to 1, they should also check whether this fact's field's "client", "host," and "sequence number" have the right relation to the fields in any of the matched "current_packet" facts. This requires 400 x (number of matched "current_packet") RAM accesses. So the total number of RAM accesses operations for Rule 2 is at least 800.

Thus, if the expert system has 200 rules similar to Rule 1 and 200 rules similar to Rule 2, the total number of search operations is at least 200*400 +200*800=240,000

Performance: Assuming a RAM's access time for a read is 10ns, one match in a Match-Select-Act cycle would take 2.4 ms without considering CPU time. This performance allows expert systems to process network traffic only at megabits/second wire speed. It is insufficient for gigabit traffic.

Conclusion: This method is straightforward. However, it is time consuming. For example, one match in a Match-Select-Act cycle would take 2.4ms without considering CPU time. This performance allows expert systems to process network traffic only at megabits/second wire speed. It is insufficient for gigabit traffic.

2.3.2 Brute Force Algorithm with facts searching the rule base

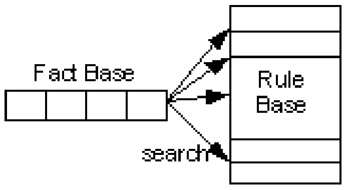
Description: A Brute-Force Matching Algorithm with facts searching the rule base [60] is shown in Figure 2.7. It is a hardware CAM-based solution. In this algorithm, all facts in the fact base are assembled into one vector and put in a register. A CAM is used as the rule base with each entry filled with one rule, in the same format of the fact vector register. The fact vector searches the rule base by searching the CAM to compare with all rules concurrently.

Example: We still use the rules in Figure 2.6 as an example. Rule 1 and Rule 2 are put in the CAM entry 1 and entry 2 shown in Figure 2.7. The current facts are shown in the fact base.

With the current facts in the fact base, the fact vector register searches the rule base CAM. The search will return a match with CAM entry 2, indicating Rule 2 is matched. Thus, if there are 400 rules in the rule base, and a CAM's parallel search takes 50 ns, this match process takes 50ns – a significant improvement over software-based solutions.

However, this method has two serious limitations:

First, it provides only a subset of functions required in the expert system's match process. It does nothing about the variable binding among multiple facts, which is an important and usually time-consuming feature of expert systems. In our example, Rule 2 implemented in CAM does nothing about Condition 3 *(TCP_State.Client = Current_Packet.DIP&Port) and (TCP_State.Host =Current_Packet.DIP&Port) and (TCP_State.Seq = Current_Packet.Ack-1).* In fact, Rule 2 should not be matched because its first TCP_state fact's sequence number doesn't equal the current packet's acknowledge number minus 1. Thus this match process is functionally deficient and will generate incorrect results.



Fact Base

Type	Packet					TCP State				TCP State					TCP State			
Field	SIP& Port	DIP& Port	Protocol	ACK#	Flag	Client	Host	State	SEQ#	Client	Host	State	SEQ#	:: ::	Client	Host	State	SEQ#
Value	A	B	TCP	3567	SYN ACK	A	B	1	1234	A	E	3	3222		C	B	3	9999

Search

Rule Base (Implemented in a CAM) **Width Grows With More Facts**

Type	Packet					TCP State				TCP State					TCP State			
Field	SIP& Port	DIP& Port	Protocol	ACK#	Flag	Client	Host	State	SEQ#	Client	Host	Stat e	SEQ#	:: ::	Client	Host	State	SEQ#
Value	X	X	TCP	X	SYN	X	X	X	X	X	X	X	X		X	X	X	X
Value	X	X	TCP	X	SYN ACK	X	X	1	X	X	X	X	X		X	X	X	X
Value									: : : :									
Value	C	D	UDP	X	X	X	X	X	X	X	X	X	X	X	X	x	X	X

Figure 2.7 A Brute Force Algorithm with Facts Searching Rule Base [60]

Second, this solution requires a fixed format of the fact base and CAM entry. This requires a very wide CAM with hundreds of facts in the fact base. More facts lead to a growth of CAM width that is limited by the current technology (e.g. the available width of a IDT CAM is 600 bits). For example, in the fact base in Figure 2.7, each TCP connection requires 32b (source_ip)+16b (source_port) + 32b (destination_ip) + 16b (destination port) + 32b (sequence number) + 4b (state number) = 132 bit. If the system has 20 TCP connection facts in the fact base, it requires a 2640 bit-wide CAM to trace these 20 TCP connections. Tracing 100 TCP connections facts requires a 13200 bit-wide CAM. This width requirement is not feasible.

Performance: If a CAM's parallel search takes 50 ns, the match process takes 50ns, no matter how many rules are in the rule base.

Conclusion: This algorithm can achieve a great speed (e.g., 50 ns for a match), but has function limitations and is not scalable with additional facts. Therefore, this CAM solution can only be applied to small and simple applications, but not to network IPS.

2.3.3 Rete-Like Matching Algorithm: Only Changed Facts Search the Rule Base

Description: Figure 2.8 shows a Rete matching algorithm with only changed facts searching the rule base. It is a software-based solution. This method is based on the observation that only a small percentage of facts (commonly <0.5% [53]) change after each Match-Select-Act cycle. Therefore a match phase processing changed facts would cost much less time than processing all facts in the entire fact base. There are distributed memories (shown as diamonds in Figure 2.8) to save the pattern matching result of unchanged facts in order to enable new changed facts to be combined with them. Besides Rete, Treat and Oflazers' Algorithms all belong to Rete-like methods [51].

Example: The same two rules are used here as an example. Rule 1 and Rule 2 are compiled to a Rete Network, which consists of two parts:

A pattern network (also called an Alpha network) with one-input alpha nodes (the circles in Figure 2.8) that tests for individual facts. In this example, Rule 1's condition is related to a one fact "current packet", so it is tested in Alpha node. Rule 2's Condition 1 is also related to a one fact "current_packet," so it is tested in Alpha node as well. Rule 2's Condition 2 is related to a one fact "TCP_state" (state=1), so this part can be tested in Alpha node. There is also alpha memory in the pattern network. It is distributed memory recording matched facts from Alpha node.

A join network (also called Beta Network) with two-input Beta nodes (the ovals in Figure 2.8), that tests the join relationship between two facts. In this example, Rule 2's Condition 3 *(TCP_State.Client=Current_Packet.DIP&Port), (TCP_State.Host =Current_Packet.DIP&Port), (TCP_State.Seq=Current_Packet.Ack-1)* is related to two facts "TCP_State" and "Current_Packet". Therefore, this testing is done in a Beta node. A Beta memory is allocated for each Beta node. The Beta memory has a left table storing all matched facts coming from the left side of Beta node and a right table storing all matched facts coming from the right side of

Beta node. The beta node tests the condition by joining these two tables. The join result is stored in Beta memory's final table. In this example, if a new TCP packet (with SYN ACK set) comes in, it will flow to the left side of the Beta node and be stored in the left table. The network already has many TCP connections in the 1st step, so TCP_State fact *a*-*c* is stored in Beta memory's right table, as was shown in Figure 2.8. The new packet in the left table will be joined with the right table according to the key specified in the Beta node. If there is any join result satisfying the Beta node conditions, the result will be stored in the final table. This finishes one match process and fires Rule 2.

Alternative Rete-like matching algorithms have been created by varying the number of state-saving intermediate memory in the join networks [53, 57]. The Treat algorithm [55] has Alpha memory but no Beta memory in its join network. This requires a re-computation of joining within the join network every Match-Select-Act cycle [53, 57]. Oflazer suggested saving the match results for every possible combination among Alpha nodes [56].

○	Alpha Node of rule base (Pattern Match)
⬭	Beta Node of rule base (Join)
◆	Intermediate Memory Storing Partial Match Result

------The Condition Part in Two Rules-------

To detect step 1 and 2 in the 3-way handshake communication in a TCP connection establishment

Rule 1:
IF *Current_Packet (Protocol =TCP) and (Flag= SYN)*

Rule 2:
IF *Current_Packet (Protocol =TCP) and (Flag=SYN_ACK)*
-- Condition 1
and *TCP_State (State =1)*
-- Condition 2
and *(TCP_State.Client = Current_Packet.SIP&Port)*
and *(TCP_State.Host = Current_Packet.DIP&Port)*
and *(TCP_state. Seq = Current_Packet.Ack-1);*
-- Condition 3

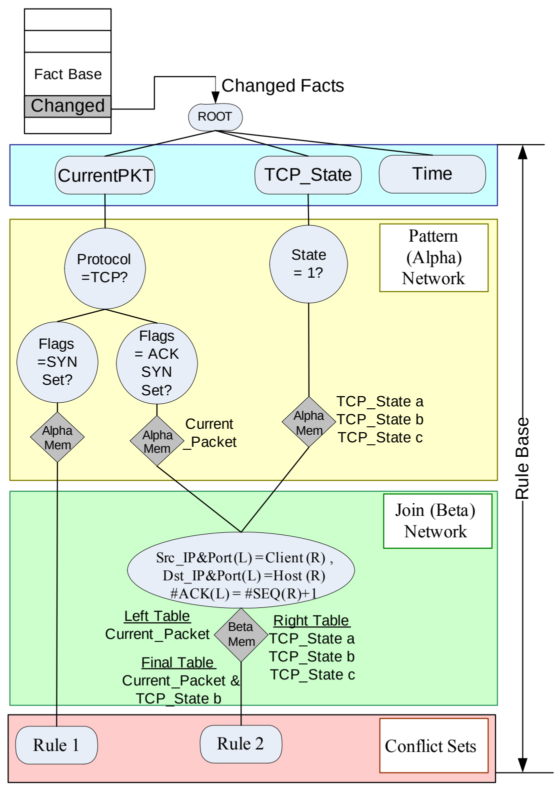


Figure 2.8 Rete Algorithm [54]

Performance: On a Sun Sparc 4 Station, an optimized Rete-based CLIPS/R2 [75] system executes 3075 changed facts [73] in 15.85s on a Manners benchmark [75] (with parameter guests =64). On another Waltz benchmark, the CLIPS/R2 can execute 12420 changed facts [73] in 5.6s [75], 24640 changed facts [73] in 12.1s [75], and 35920 changed facts [73] in 18.6s [75], respectively (with parameters regions =12, 25, 37 respectively). This means one Match-Select-Act cycle takes 0.45ms-5ms in different applications for a CLIPS/R2 system.

On a Pentium III CPU, an optimized Rete-based OPSJ [76] executes the same benchmark in 1.6s, 0.4s, 0.9s, and 1.5s [76]. This means one Match-Select-Act cycle takes 0.03ms-0.5ms in different applications for an OPSJ system.

This performance allows expert systems to process network traffic only at ten-megabits/second wire speed. This is insufficient for gigabit traffic.

Conclusion: By only processing the changed facts, this algorithm performs better than the Brute Force Algorithm with rules searching the fact base. One Match-Select-Act cycle takes 30us-5ms in this method. It has been widely used in many expert systems. However, this performance allows the system to process network traffic only at ten-megabits/second wire speed, and it is yet insufficient for gigabit traffic.

2.3.4 Multi-Processor Solutions for the Match Phase

A large amount of work has been done to exploit parallelisms in the match phase of Rete-like expert systems by distributing the workload onto multiple processors.

Description: A multiprocessor executes a Match-Select-Act cycle by dividing the whole rule set into a number of subsets and assigning them to multiple concurrent processors working as "Match" Processing Elements [51-53]. The related fact base is copied to each match processing element as a local fact base. Match processing elements perform parallel matching and thus produce all matched rules that are combined as the final conflict set. A separate processor working as a "Select" processing element selects one rule from all matched rules to fire [52]. It also notifies each match processing element about its decision through buses so the related local fact base in match processing element gets updated [52]. This finishes a Match-Select-Act cycle in a multi-processor system.

Examples: Various multi-processor systems have been explored. These variations include: different numbers of PEs, from tens [51, 53] to hundreds to tens of thousands [58, 59]; different grains of rule partition, including nodes inside a Rete network (e.g, PSM [51, 53]), one rule, and a set of multiple rules (e.g., DADO, NON-VON [58, 59]); and different architectural systems, including shared-bus structures with similar processing elements [51] and tree-based structures with multiple levels of processing elements [58,59].

Performance: Gupta and Forgy concluded that limited parallelism existed in the match phase as only about a ten-fold improvement can be achieved by using extremely fine grain parallelism in the match phase [51]. This means by parallelizing the match phase, one Match-Select-Act cycle in multi-processor systems would take 3us-500us, if they used the same algorithm and benchmark discussed in Section 2.2.4. This performance allows expert systems to process network traffic only at hundred-megabits/second wire speed and is therefore insufficient for gigabit traffic.

Conclusions: Parallel processing speeds up the computationally intensive match process. However, it calls for more complicated and costly implementation than the uni-processor method. Its speedup is limited (e.g., 3 us for a Match-Select-Act cycle). This performance allows expert systems to process network traffic only at hundred-megabits/second wire speed and is therfore insufficient for gigabit traffic.

2.3.5 Content Addressable Memory Accelerated Multi-Processor System

Content Addressable Memory has been proposed to help accelerate Multiprocessor expert systems in two ways [63-66]. Both methods employ it inside processing elements of a Rete-based multiprocessor system.

The first method [63-64] uses CAM in a Distributed Rete [64] multi-processor system. Each match processing element consists of a processor, a RAM and a CAM, and it processes several nodes in the Rete network. The CAM stores node identifier information for the nodes processed in this processing element. Therefore, by searching the CAM, a processing element can decide whether it should accept and process or ignore an incoming fact. The CUPID system is a match multi-processor implemented based on this method. A CUPID architecture with 64 processors will improve performance by 10 times over a uni-processor system with the same CPU performance. This brings similar performance to the mult-systems discussed in Section 2.4 to process hundreds of megabits/second traffic.

The second method [65-66] uses CAM to accelerate control and communication mechanisms inside the multi-processor system. In this method, each processing element performs its own select and act processes, if possible in parallel with the match process. CAM is used as the control mechanism in each processing element to execute select and act phases correctly before the match process finishes. CAM also helps communications between each processing element and other processing elements. The average CAM-assisted performance improvement in this architecture is 25 in various benchmarks, which allows one Match-Select-Act cycle to execute in 1.2us -20us. This performance allows expert systems to process network traffic only at hundred-megabits/second wire speed.

Conclusions: Both methods accelerate control and communication mechanisms in a multi-processor system [63-66]. Neither method applies CAM directly to the match process. Both have brought performance improvement (e.g., one Match-Select-Act cycle takes time in us range). However, this performance allows expert systems to process network traffic only at hundred-megabits/second wire speed, which is not sufficient for gigabit IPS.

3.0 A UNIQUE HYBRID ENCODING SCHEME FOR EFFICIENT RANGE MATCHING IN TERNARY CONTENT ADDRESSABLE MEMORY

The rapid rate of growth in the Internet has created a great deal of interest in expanding network transmission bandwidth and increasing transmission speed. While these developments are exciting, they have brought another problem in that the amount of time available for a network device (such as a switch) to respond to a single packet has decreased. This requires network devices to be accelerated and have faster processing speed.

Content Addressable Memory (CAM) [78] is a parallel search engine chip that searches its memory content in parallel to find one or more entries that are equal to the input data. It has been proposed to accelerate many network functions such as packet classification and intrusion detection that are search-intensive by nature [79, 80]. Compared to software-based network function approaches, ternary CAMs offer many benefits. Unlike software-based solutions that perform serial searches and cost linear search time, CAM performs a search operation in parallel and takes a constant number of clock cycles, typically tens of nanoseconds in current technology. This is 100 times to 1000 times faster than software running on processors. In addition, the search time is independent of the number of rules used in a given search. This consistently high performance accounts for its popularity in gigabit network applications.

Besides an exact matching function which determines if the input data equals any memory content, a range matching function to determine if the input data are within a range is also required in many network functions. Though ternary CAM provides great performance improvement in single-cycle exact matching, it does have a serious limitation in single-cycle range matching. Single-cycle range matching using existing encoding schemes including binary code, one-hot encoding, two-hot encoding, Johnson coding and Gray coding exhibits scalability issues in CAM size, either in the number of CAM entries required (depth expansion) or the number of bits required to represent a specified number (width expansion).

We propose a unique encoding scheme which fits into an existing commercial CAM to perform single-cycle range matching more efficiently. This hybrid encoding scheme mixes the features of one-hot codes and Johnson codes. To perform a range matching using a single normal TCAM search operation, we will encode the range to be matched and the input data using the encoding scheme. This novel encoding scheme provides a method to do the single-cycle range matching in CAM with limited depth and width expansion. It is the foundation of hardware acceleration of various network functions, including the IDS function where many range matching criteria exist in IDS rules.

3.1 SURVEY OF EXISTING SOLUTIONS OF SINGLE-CYCLE CAM RANGE MATCHING

3.1.1 Single-Cycle Range Matching Using Ternary CAMs

Content Addressable Memory (CAM) [78] is a parallel search engine which searches its memory content in parallel to find one or more equals to the input data (called search key here). A ternary CAM can have 3 values in one bit: ‘0’, ‘1’, and “don’t care (X)”. A “don’t care” bit will match either ‘1’ or ‘0’ bit values in the search key. In this thesis, “CAM” is used to indicate commercially available Ternary CAMs. To be accurate, let us define the functions that a “CAM” usually have and doesn't have, listed as follows:

A CAM does have functions of:

- Single match to find the first data entry in it that matches the input data
- Multi-match to find all data entries that match the input data, with matched resulted pulled out one by one, serially by scroll operations.
- Writing to one CAM entry, just as a RAM write function
- Invalidating all the entries that matches the input data in parallel
- Learning a new data entry if this data is not in the CAM

A CAM doesn’t have functions of:

- Counting the number of matched results in a Multi-Match operation

- Bit vectors to show Multi-match results in parallel in single cycle
- Bit serial comparison when matching
- Parallel writing

A commercially available CAM IDT network search engine IDT75K62100 CAM [85] is used in this research. It has total 9Mbits with each entry configurable as 72 bits wide, 144 bits wide, 288 bits wide or the longest 576 bits wide. Internal clock is 10ns wide and 100Mhz. Single search operation can be issued in pipeline. It can achieve an input processing speed of 14.4Gbps.

Exact matching to determine if the input data equals to any memory content can be performed in parallel using Ternary CAM in a single cycle. For example, as shown in Figure 3.1, key "0X10" will be compared in parallel to 5 numbers stored in CAM entries. It will match CAM entries "X010" and "011X".

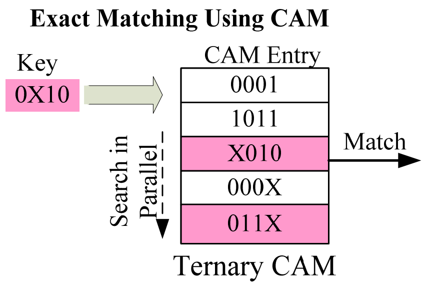


Figure 3.1 Exact Matching Using CAM

To implement single-cycle range matching using CAM, this range criterion must be expanded to multiple CAM entries to cover all numbers in a specified range. A match to any of these entries indicates the data word is within the specified range. For example, Figure 3.2 shows a CAM search operation for a range matching to test whether 27 matches the range 4-61. The data range 4-61 is encoded with binary code and requires 7 CAM entries: 00,01XX including numbers 4 through 7, 00,1XXX including numbers 8 through 15, 01,XXXX including numbers 16 through 31, 10,XXXX including numbers 32 though 47, 11,0XXX including numbers 48 through 55, 11,10XX including numbers 56 through 59, and 11,110X including numbers 60 and 61. The input key of 27 (01,1011) will match the 3rd entry 0001,XXXX, indicating that 27 belongs to the range 4-130. This range matching can be done by one search operation in TCAM, but requires 7 CAM entries to represent the range.

In general, when using binary encoding, the number of entries required to represent the range is $O(2N\text{-}2)$ where N is the data width. If one rule contains multiple range criteria for different fields of the packet, the total number of entries is $O((2N\text{-}2)^R)$, where R is the number of multiple range criteria. For example, a single rule with two range criteria both in 16-bit fields would result in as many as 30^2 or 900 entries in a CAM. This lack of depth scalability is problematic for CAM space and rule updates, making it impractical for many real applications. In addition, updating a rule may require writing hundreds of CAM entries, making it impractical for many real time applications.

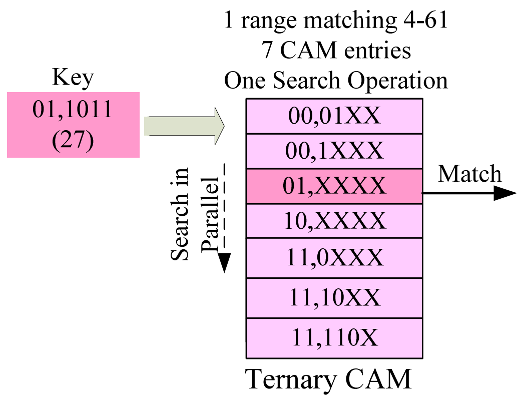


Figure 3.2 Range Matching Using CAM with Binary Code

3.1.2 Performance Metrics

To survey existing encoding schemes for potential use in CAM for range matching with a normal search operation, two performance metrics are important in the encoding scheme that will affect the efficiency of performing range matching in a CAM:

1. The number of bits required to represent a specified code (width expansion) should be as small as possible, in order to lower the width requirement of a CAM. That means a given width of data should be able to accommodate as many data as possible. Denser encoding schemes generally will achieve this goal more successfully. For example, binary encoding provides the maximum achievable efficiency in encoding specified data.

2. The number of CAM entries required to represent an arbitrary range (depth expansion) should be as small as possible, preferably a constant number rather than a number linear with the data width N, in order to lower the depth requirement of a CAM and minimize the updating time.

Sparser encoding schemes generally meet this goal more successfully. For example, one-hot encoding allows an arbitrary range of values to be specified with a single CAM entry.

Unfortunately, the two goals are mutually exclusive. An encoding scheme with better code capacity will consequently require more CAM entries to map a specified range of values. In the worst case, an entry for each discrete value in the range may be required. In contrast, an encoding scheme that requires fewer CAM entries achieves that advantage usually by expanding width, reducing the code capacity. The challenge, therefore, is to trade off these mutually exclusive goals to find the optimal compromise.

Thus in this research we are interested in two encoding scheme performance metrics:

(1) Encoding efficiency represented by code capacity. This decides the width expansion in a CAM for the range matching problem.

(2) The maximum number of CAM entries required for an arbitrary range matching. This determines the depth expansion in a CAM for the range matching problem.

3.1.3 Existing Encoding Schemes

We surveyed various encoding schemes including binary, gray, one-hot, two-hot, and Johnson code, as potential candidates for TCAM based range matching. Two performance metrics are analyzed below, and summarized in Figure 3.3.

- **Binary code:**

For an N bit binary code, the maximum number of entries required is dependant on N. In the example given in Figure 3.3, N-1 entries to to cover all possible numbers starting with '0' are needed to represent anything between 000..001 to 111..110. These entries are: 000..001, 000..01X, 000..1XX, until 01X...XXX. We also need N-1 entries to cover all possible numbers starting with '1'. These are 111..110, 111..10X, 111..0XX, 110..XXX, until 10X..XXX. Therefore the maximum number of entries required for an arbitrary range is 2N-2. An N bit binary code can accommodate 2^N distinct numbers.

- **Gray code:**

Gray code is very similar to Binary code except any adjacent code words differ in only one bit. According to the most significant 2 bits in gray code, all code words can be separated into 4 divisions. N-2 entries are needed to cover any range inside one division. One or two complete divisions can be represented in one entry because their headers only differ in 1 bit which can be "X (don't care)" in CAM. So an arbitrary range can be represented in N-2 +1 + N-2 =2N-3 CAM entries. An N bit Gray code can accommodate 2^N distinct numbers.

- **One-hot encoding**

Any code word in one-hot encoding only contains a single non-zero bit. It can be expressed in a regular expression 0*10*. The '1' bit migrates from the right most position to the left most position as the number increases. This feature allows it to represent any range in a single CAM entry, with "00..0" at the beginning to specify the "less than" case and "00..0" at the end specifying the "greater than" case. However, an N bit one-hot code can only accommodate N distinctive numbers.

- **Two-hot encoding**

Code words in two-hot encoding always have two bits of '1'. It can be expressed as 0*10*10*. According to the different positions of the high '1', code words can be divided into N-1 divisions. Within each division, the lower '1' bit is encoded as "one-hot" encoding. Therefore a single CAM entry can represent any range inside one division. Division 0 only has one number and it can be combined with division 1 if both are included in the range. Therefore a range across (N-1) divisions will require N-2 CAM entries. An N bit two-hot code can accommodate $(\sum_{1}^{N-1} i)$ = N*(N-1)/2 distinctive numbers.

- **Johnson code**

Johnson code works similar to one-hot encoding except that it has multiple '1's and two divisions with different most significant bits. It can be expressed in a regular expression 0*1*0*. A single entry can cover the "less than" or the "greater than" case within one division.

Therefore the maximum number of entries required is 2. An N bit one-hot code can accommodate 2N distinctive numbers.

In conclusion, Gray and two-hot encodings exhibit poor depth scalability similar to binary code as the number of required CAM entries increases linearly with *N*. One-hot and Johnson encoding have good depth scalability requiring a constant number of entries irrespective of *N*. Unfortunately, both one-hot and Johnson coding exhibit poor width scalability, as the data width required for a code word grows linearly (*O(M)*) with the magnitude (*M*) of the maximum number in a range. The width of TCAM, about 600 bits with current technology, is quickly exceeded. Therefore, existing encoding schemes exhibit either poor depth or width scalability for range matching.

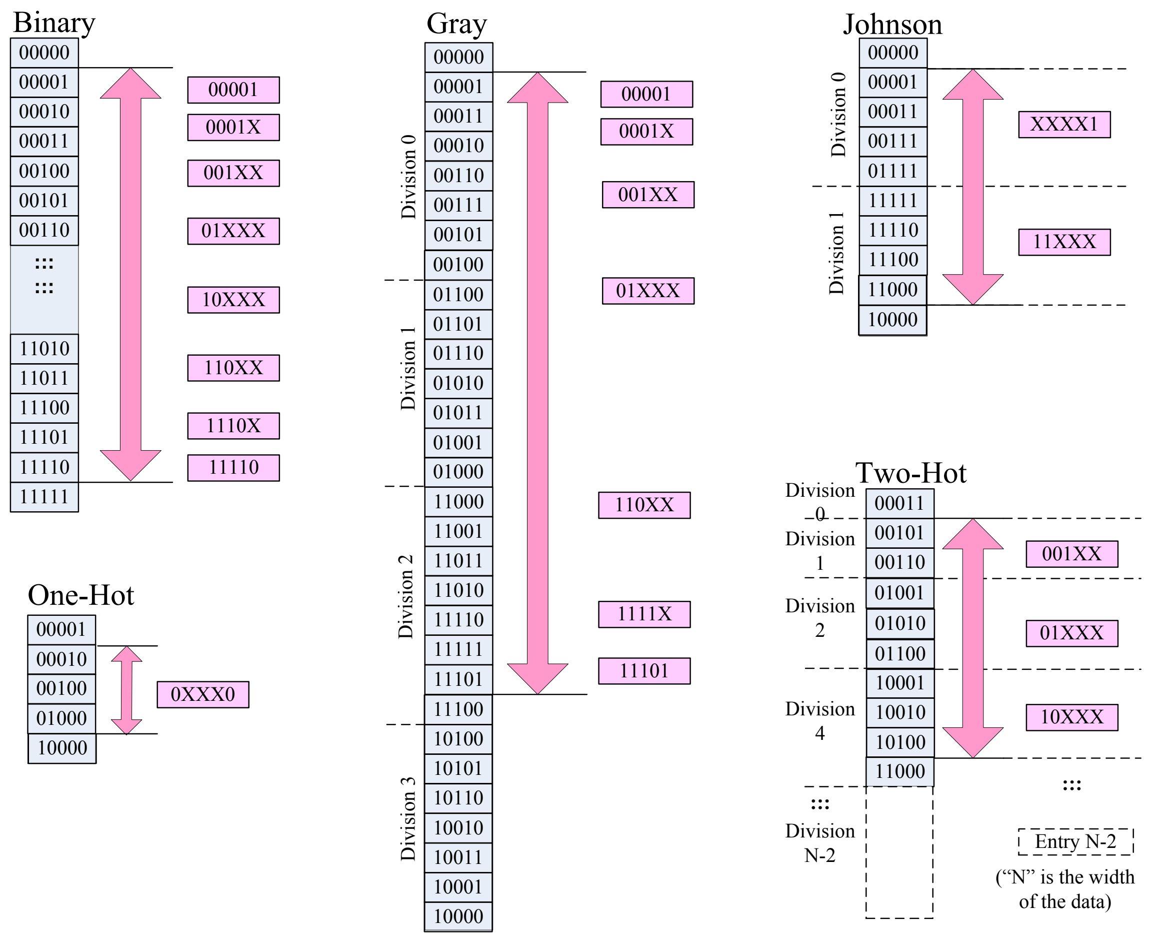


Figure 3.3 Analysis of Number of CAM Entries Required for a Range Matching in Different Coding

3.1.4 Other Solutions

Methods other than encoding scheme have been proposed to solve the range matching problem in a CAM. Liu [81] uses a lookup table (LUT) to translate each distinctive range into one bit in one-hot style, requiring only a single CAM entry for an arbitrary range. However, this technique depends on the assumption that very few ranges will be employed. In the worst case, the width scalability of $(M^2 - M)/2$ exceeds that of one-hot encoding M. Other disadvantages include additional hardware required besides CAM, and the need to update all entries in the LUT and CAM when adding a new range to the system.

This thesis proposes a hybrid encoding scheme mixing the features of one-hot and Johnson codes to perform range matching more efficiently within a TCAM. The proposed encoding scheme represents an arbitrary range with a maximum of three CAM entries while providing better width scalability $O(\sqrt{M})$ than one-hot and Johnson encoding. Range matching using this encoding scheme can be done through a single constant-time search operation.

3.2 DEFINITION OF CODE WORDS OF A NOVEL ENCODING SCHEME

A unique hybrid encoding scheme was developed to fit into an existing commercial CAM to perform single-cycle range matching more efficiently. To perform a range matching using one normal TCAM search operation, we will encode the range to be matched and the input data using the encoding scheme, with a limited CAM size expansion. For example, in Figure 3.4, the

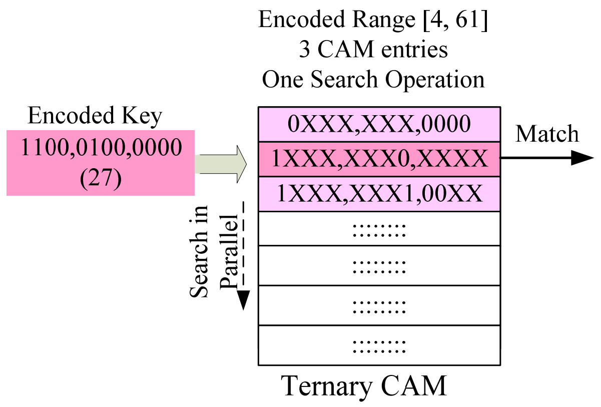


Figure 3.4 Using Range Matching Using CAM With the Hybrid Encoding Scheme

key 27 and the range 4-61 are both encoded by this hybrid scheme, with only 3 CAM entries and 12 bits in width. The range matching can be done in one search operation and the range is represented in only 3 CAM entries.

This encoding approach mixes aspects of one-hot and Johnson encoding, resulting in a coding scheme that is more efficient than any single code mentioned in Section 3.1.

3.2.1 Regular Expression

A typical code word in this hybrid encoding scheme is shown in Figure 3.5. It can also be represented in a regular expression 1*00*10*.

Each code word has two parts: *header* 1*0 and *body* 0*10*. An asterisk (*) following a digit indicates that the digit occurs zero or more times. The variable length *header* consists of a sequence of contiguous '1' bits called *header* '1's (e.g. 1*), and terminates with a single '0' bit called the *header* '0.' The *body* consists of a single'1' bit called *body* '1,' sitting between two fields of contiguous '0' bits (0*). The higher field is called *clamped* '0's and the lower field called *tail* '0's.

Consider an *N*-bit code word with an *H*-bit *header* and an *N-H* bit *body* which has *T* bits of *tail* '0's. This word may be expressed as $1^{H-1}00^{N-H-T-1}10^{T}$, where *H* can grow from 1 to *N*-1, and *T* can grow from 0 to *N-H.*

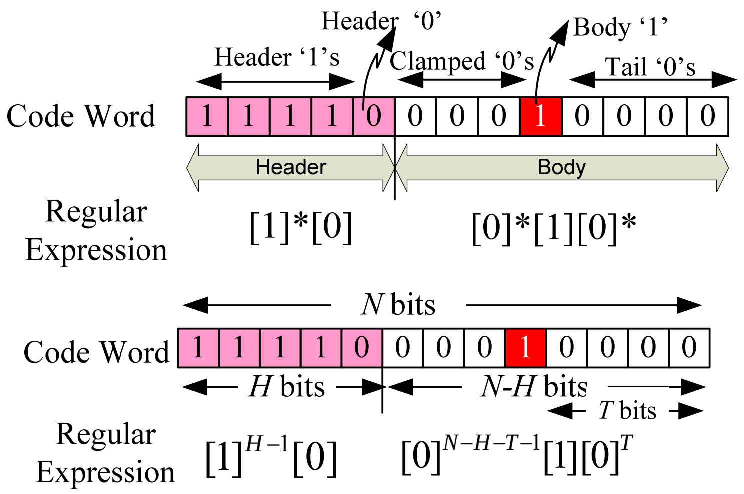


Figure 3.5 A Typical Code Word Consists of a Header and a Body

3.2.2 Generating Algorithm

Integers 0 through N*(N-1)/2 can be encoded into N-bit code words through a simple generating algorithm shown as follows, where Wi is the encoded code word for an integer i.

```
i=0;
For (H=1; H++; H<=N-1)
{
For (T=0; T++, i++; T<=N-H-1)
        Wi=$[1]^{H-1}[0][0]^{N-H-T-1}[1][0]^{T}$
    }
End
```

All words that have the same *header* make up one division, called "division *H*-1." Therefore, iteration of the inner-most loop with a constant H generates all Wi's belonging to division H-1. Processing the outside loop generates Wi for different divisions 0 through N-2, with different headers.

Figure 3.6 shows an example of encoded number range 0 to 66 in 12 bit code words using this hybrid encoding scheme. These code words are separated into different divisions 0-10 according to the header value. For example, numbers 0 to 11 with header "0" and H=1 belong to division 0; numbers 11 to 20 with header "10" and H=2 belong to division 1; Division 10 has only one number 66, featured with header "111111111110" and H=11.

Inside each division, *bodies* of all the code words are one-hot encoded. The *body* '1' migrates from the right-most bit to the left-most bit of the *body* as number increases, as shown in Figure 3.6. When the *body* '1' reaches the left-most position in the *body*, the current division ends and the next division begins, featuring a longer *header* and a shorter *body*, as shown in Figure 3.6. In general, the process of one-hot encoding inside the *body* is repeated for each division, while the *header* grows with one more bit of *header* '1' from division to division. This process continues until the number of bits in the *body* reduces to 1, and the *header* grows to N-1 bits, yielding the last code word "1111…101". An *N* bit code can accommodate *N**(*N*-1)/2 distinct code words.

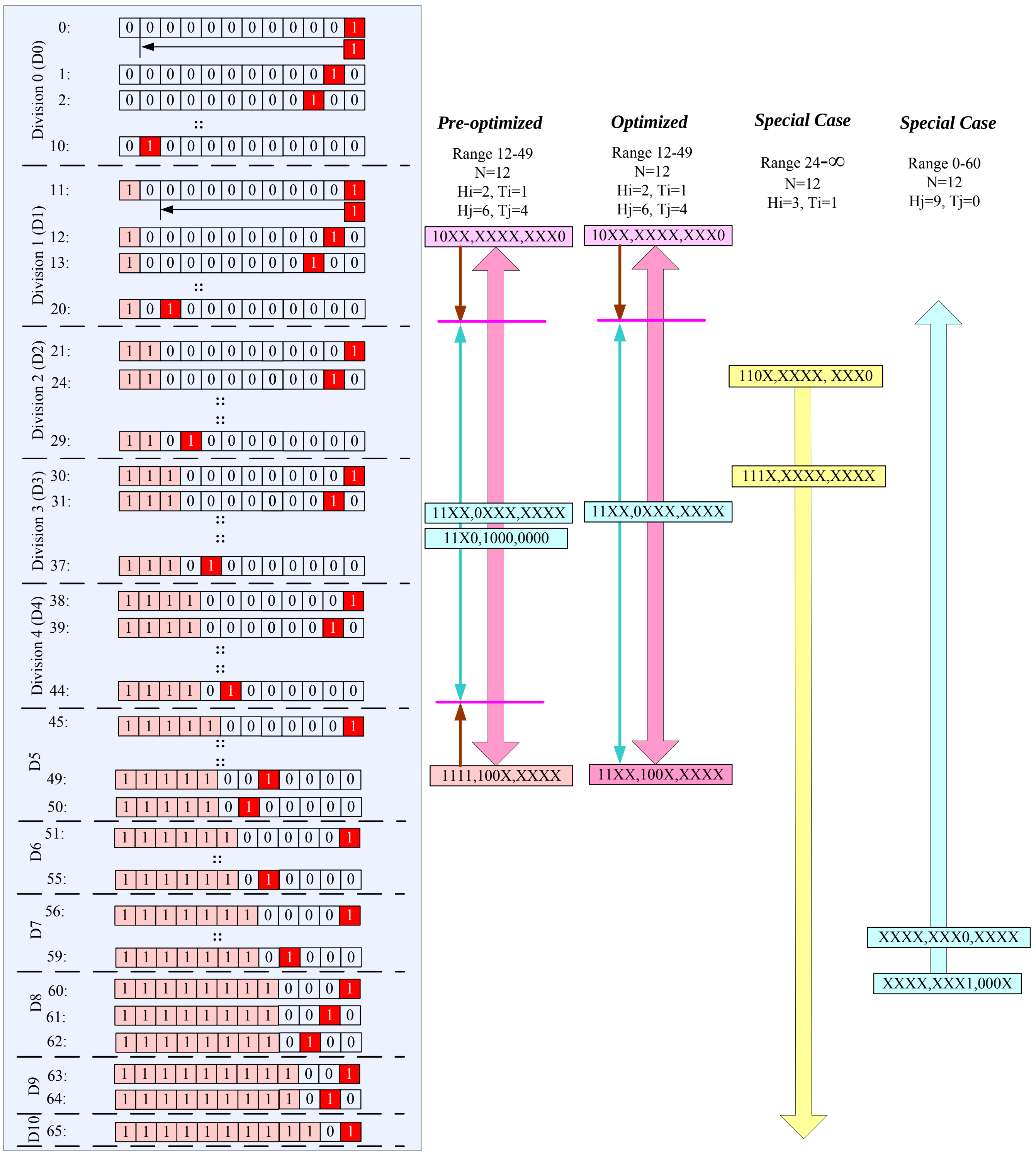


Figure 3.6 Example of 12 Bit Code Words Encoded for Number 0 Through 66

3.2.3 Encoding and Decoding Algorithms

The formula of encoding and decoding of this hybrid encoding scheme is deduced as follows:

To encode an integer i into an N-bit code word Wi (i< N*(N-1)/2):

Wi=F(N, i) $=[1]^{X-1}[0][0]^{N-X-Y-1}[1][0]^{Y}$

where X is the maximum integer which satisfies the equation $\sum_{x=1}^{X-1}(N-x) < i+1$ =>

$$X^2-(2N-1)X+2i+2>0 \Rightarrow X = N-\left\lceil \frac{\sqrt{(2N-1)^2-8i-8}-1}{2} \right\rceil$$

$$\text{Y}=i-\sum_{x=1}^{X-1}(N-x)=i-(X-1)(N-X-1)$$

$$= i-\frac{(N-\left\lceil \frac{\sqrt{(2N-1)^2-8i-8}+1}{2} \right\rceil)*(N+\left\lceil \frac{\sqrt{(2N-1)^2-8i-8}-1}{2} \right\rceil)}{2}$$

Therefore, an integer *i* can be encoded into *Wi*=F*(N, i)* =

$$[1]^{N-\left\lceil \frac{\sqrt{(2N-1)^2-8i-8}+1}{2} \right\rceil}[0][0]^{\left\lceil \frac{\sqrt{(2N-1)^2-8i-8}-1}{2} \right\rceil-i+\frac{(N-\left\lceil \frac{\sqrt{(2N-1)^2-8i-8}+1}{2} \right\rceil)*(N+\left\lceil \frac{\sqrt{(2N-1)^2-8i-8}-1}{2} \right\rceil)}{2}}[1][0]^{i-\frac{(N-\left\lceil \frac{\sqrt{(2N-1)^2-8i-8}+1}{2} \right\rceil)*(N+\left\lceil \frac{\sqrt{(2N-1)^2-8i-8}-1}{2} \right\rceil)}{2}}$$

To decode a code word Wi back to integer i

Assume Wi=$[1]^{H-1}[0][0]^{N-H-T-1}[1][0]^{T}$

$$\text{i = f(H, N, T)} = \sum_{m=1}^{H-1}(N-m)+T = N*(H-1)-(H-1)*(H-2)/2+T$$

$$= NH-N-\frac{H^2-3H}{2}+T-1$$

This encoding and decoding algorithm can be implemented in a Look-Up Table (LUT), requiring one clock cycle to get the encoded or decoded result.

3.3 MAPPING AN ARBITRARY RANGE INTO CAM

The primary benefit of this coding scheme is that it allows an arbitrary range to be represented in a maximum of three CAM entries with a higher width density than other constant depth scaling encodings. The following three steps explain mapping a range *i-j* into a TCAM using three entries:

3.3.1 Step 1: Encode i and j into code word Wi and Wj:

Wi= F(N, i) $=[1]^{H_i-1}[0][0]^{N-H_i-T_i-1}[1][0]^{T_i}$

Wj= F(N, j) $=[1]^{H_j-1}[0][0]^{N-H_j-T_j-1}[1][0]^{T_j}$

where

$$H_i = N - \left\lceil \frac{\sqrt{(2N-1)^2-8i-8}-1}{2} \right\rceil$$

$$T_i =$$

$$i - \frac{(N - \left\lceil \frac{\sqrt{(2N-1)^2-8i-8}+1}{2} \right\rceil)*(N + \left\lceil \frac{\sqrt{(2N-1)^2-8i-8}-1}{2} \right\rceil)}{2}$$

$$H_j = N - \left\lceil \frac{\sqrt{(2N-1)^2-8j-8}-1}{2} \right\rceil$$

$$T_j =$$

$$j - \frac{(N - \left\lceil \frac{\sqrt{(2N-1)^2-8j-8}+1}{2} \right\rceil)*(N + \left\lceil \frac{\sqrt{(2N-1)^2-8j-8}-1}{2} \right\rceil)}{2}$$

3.3.2 Step 2: Calculate the pre-optimized four CAM entries that represent the range Wi to Wj

An algorithm to calculate the pre-optimized four CAM entries that represent the range Wi to Wj is shown in Figure 3.7.

To match all possible values within the range without matching others outside the range, the matching is logically partitioned into four entries. The first entry matches all numbers equal to or greater than W_i inside W_i's division. Similarly, the last entry (entry 4) matches all number equal to or less than W_j inside W_j's division. Entries 2 and 3 match all numbers in the intermediate divisions between W_i and W_j with the N-Hj+1th bit be '0' or '1' respectively.

Entry 1, $1^{Hi-1}0X^{N-Hi-Ti}0^{Ti}$, has the same header as Wi, restricting the coverage to the same division. It copies the tail '0's of Wi and fills the other bits with "X (don't care)". The length of tail '0's excludes any words smaller than Wi in this division. This is true because analogous to one-hot encoding, the body '1' migrates toward the most significant bit, which causes the length of tail '0's to increase when a number grows within the same division. Therefore, entry 1 covers exactly all code words greater than or equal to Wi inside Wi's division, but excludes smaller numbers.

Entry 4, $1^{Hj-1}00^{N-Hj-Tj-1}X^{Tj+1}$, has the same header as Wj, restricting the coverage to the same division. It copies the clamped '0's of Wj and fills the other bits with "X (don't care)". The length of the clamped '0's excludes any words greater than Wj in this division. This occurs because analogous to one-hot encoding, the body '1' migrates toward the most significant bit, which causes the length of clamped '0's to shrink when a number grows within the same division. Therefore, entry 4 covers exactly all code words less than or equal to Wj within Wj's division, but excludes greater numbers.

Entry 2 $1^{Hi}X^{Hj-Hi-2}0X^{N-Hj+1}$ fills '1's into Wi's header, converts the last header '1' bit of Wj to '0' (N-Hj+1th bit) and fills other bits with "X (don't care)". It excludes all numbers in division Wi and lower because of the contiguous '1's (header '1's) from N-Hi to N bit. It.matches numbers only in the divisions between the division for W_i and W_j by specifying the *header* '1's to start with the division after W_i and using X's to allow the header to grow to match larger divisions. The '0' is placed to prevent the header from matching the division including W_j and beyond. For smaller divisions, the '0' can be either a *header '0'*, *clamped* '0' or a *tail* '0'. This covers all values in intermediate divisions except for the values where their *body* '1' conflicts with this '0' at the same position. There is a conflict in every intermediate division except the division just prior to that containing W_j. These conflicts are handled by entry 3. Therefore, entry 2 covers all numbers with a '0' at its N-Hj+1th bit in divisions between Wi's and

Wj’s. This entry is valid only when Hj-Hi>=2. If Hj-Hi<2, this entry is invalid and can be ignored.

Entry 3 $1^{Hi}X^{Hj-Hi-3}010^{N-Hj+1}$ fills ‘1’s into Wi’s header, copies the last header ‘1’of Wj (N-Hj+1th bit), puts ‘0’s into the bit before it (N-Hj+2th bit) and all bits after it, and then fills the rest bits with “X (don’t care)”. It covers the remaining intermediate values not covered by entry 2. The specified *header* ‘1’s in the entry are the same as entry 2. The specified ‘0’ is moved one position left limiting the header to match only the divisions in conflict from entry 2. Finally, the specified ‘0’ from entry 2 is replaced with a ‘1’ to match the *body* ‘1’ at the N-Hj+1th bit that

Wi $\quad [1]^{Hi-1}[0][0]^{N-Hi-Ti-1}[1][0]^{Ti}$

CAM Entry 1 $\quad [1]^{Hi-1}[0][X]^{N-Hi-Ti}[0]^{Ti}$

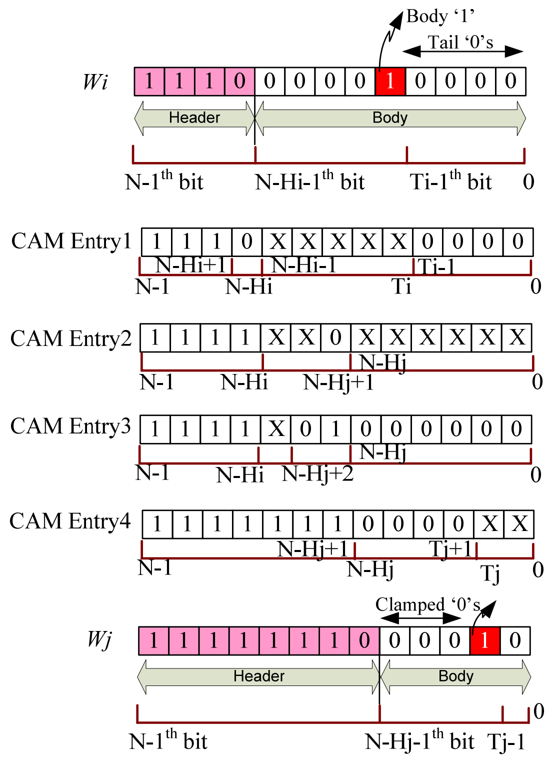
CAM Entry 2 $\quad [1]^{Hi}[X]^{Hj-Hi-2}[0][X]^{N-Hj+1}$
(Valid only when Hj-Hi-2>=0)

CAM Entry 3 $\quad [1]^{Hi}[X]^{Hj-Hi-3}[0][1][0]^{N-Hj+1}$
(Valid only when Hj-Hi-3>=0)

CAM Entry 4 $\quad [1]^{Hj-1}[0][0]^{N-Hj-Tj-1}[X]^{Tj+1}$

Wj $\quad [1]^{Hj-1}[0][0]^{N-Hj-Tj-1}[1][0]^{Tj}$

(a) The Pre-Optimized Four CAM Entries in Regular Expression



(b) The Pre-Optimized Four CAM Entries in Diagram

Figure 3.7 An Algorithm to Generate the Pre-Optimized Four CAM Entries for Range Wi to Wj

would have been in conflict from entry 2. Bits 0 to N-Hj must be tail '0's. This entry 3 covers all numbers with a '1' at its N- Hj+1th bit in all divisions between Wi's and Wj's. This entry is valid only when Hj-Hi>=3. If Hj-Hi<3, this entry is invalid and can be ignored.

Combining entry 2 and entry 3 matches exactly all numbers in divisions higher than Wi and lower than Wj, whether the N-Hj+1th bit is '0' or '1'.

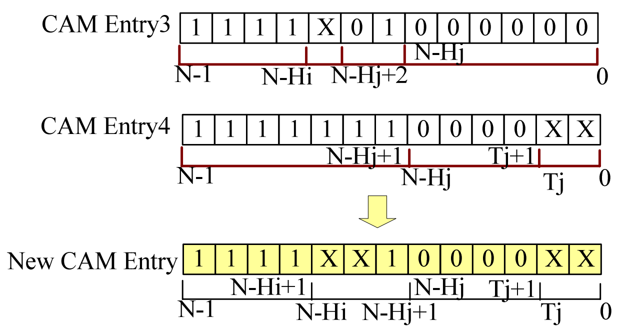
In the example shown in Figure 3.6, we want to find all 12 bit code words between 12 (1000,0000,0010) and 49 (1111,1001,0000). Entry 1, 10XX,XXXX,XXX0, matches the code words 12 (1000,0000,0010) through 20 (1010,0000,0000) in division 1. Entry 2, 11XX,0XXX,XXXX, matches code words 21 (1100,0000,0001) through 44 (1111,0100,0000), except code word 28 (1100,1000,0000) and code word 37 (1110,1000,0000). Entry 311X0,1000,0000 matches code words 28 (1100,1000,0000) and code word 37 (1110,1000,0000). Thus entry 2 and entry 3 covers division 2 though division 4. Entry 4 1111,100X,XXXX matches code word 45 (1111,1000,0001) through 49 (1111,1001,0000) in division 5. These four entries completely cover the range 12-49.

3.3.3 Step 3: Calculate the optimized three CAM entries that represent the range Wi to Wj

In this step we will optimize the four CAM entries in Step 2 to minimize the number of CAM entries needed for a range Wi to Wj.

Two of the CAM entries from Step 4.2 (entries 3 and 4) can be combined into a single new CAM entry $1^{Hi}X^{Hj-Hi-2}10^{N-Hj-Tj}X^{Tj+1}$ when the condition Hj-Hi>=2 holds, as shown in Figure 3.8. Again, like the original entry 3, the specified header '1's limit the divisions matched to those greater than that containing W_i. Like the original entry 4, the *clamped* '0's are specified to prevent values greater than W_j from being matched. The combined new CAM entry has bits Tj+1 to N-Hj+1 and bits N-Hi to N-1 exactly same as both old entry 3 and entry 4.

The key to combining this new entry is the '1' at bit N-Hj+1 towards the middle of the string. In the new entry, this '1' can either be a header '1' or a body '1'. If it is a header '1', the 'X's to the left of this '1' are all '1's, making the string behave like the original entry 4. Otherwise if this '1' is the *body* '1', it means that to match valid words the 'X' bit to the left of this '1' will be a '0' and that the far right 'X's are also '0's as part of the *tail*. This string



(a) Combine Two CAM Entries in Step 2 into One New CAM Entry

Wi $[1]^{Hi-1}[0][0]^{N-Hi-Ti-1}[1][0]^{Ti}$

CAM Entry 1 $[1]^{Hi-1}[0][X]^{N-Hi-Ti}[0]^{Ti}$

CAM Entry 2 $[1]^{Hi}[X]^{Hj-Hi-2}[0][X]^{N-Hj+1}$

(Valid only when Hj-Hi-2>=0)

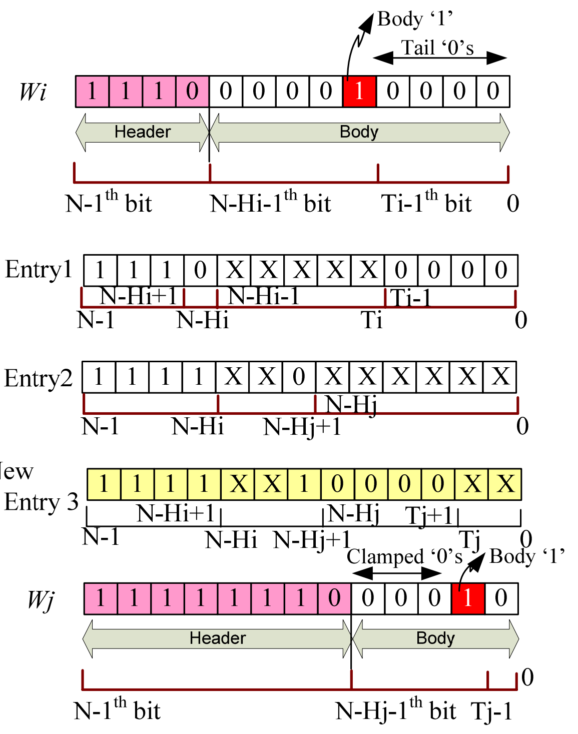
New CAM Entry 3 $[1]^{Hi}[X]^{Hj-Hi-2}[1][0]^{N-Hj-Tj}[X]^{Tj+1}$

(Valid only when Hj-Hi-2>=0)

Wj $[1]^{Hj-1}[0][0]^{N-Hj-Tj-1}[1][0]^{Tj}$

(This optimization holds true only when Hj-Hi >=2)

(b) The Optimized Three CAM Entries in Regular Expression



(c) The Optimized Three CAM Entries in Diagram

Figure 3.8 An Optimized Algorithm to Represent the Range Wi to Wj in Three CAM Entries

behaves like the original entry 3. Therefore, the new entry equals to the combination of entry 3 and entry 4. The four CAM entries in Step 2 can be optimized to only three CAM entries with exactly the same coverage when condition Hj-Hi >=2 holds.

In the example shown in Figure 3.6, we want to find all 12 bit code words between 12 (1000,0000,0010) and 49 (1111,1001,0000). Entry 1, 10XX,XXXX,XXX0, matches the code words 12 (1000,0000,0010) through 20 (1010,0000,0000) in division 1. Entry 2, 11XX,0XXX,XXXX, matches code words 21 (1100,0000,0001) through 44 (1111,0100,0000), except code word 28 (1100,1000,0000) and code word 37 (1110,1000,0000). Entry 3 11XX,100X,XXXX matches code words 28 (1100,1000,0000), code word 37 (1110,1000,0000) and code word 45 (1111,1000,0001) through 49 (1111,1001,0000). Thus these three entries completely cover the range 12-49.

Therefore, any arbitrary range in an N-bit wide code word can be represented in a maximum of three CAM entries.

3.3.4 Step 4: Special Cases

- **Wi to ∞: "Greater Than or Equal To" A Specified Code Word Wi**

The first special case is Wj=∞ in the range Wi to Wj. The range Wi to ∞ means greater than or equal to a specified code word Wi.

When Wj=∞, the biggest number in an N-bit code word is $[1]^{N-2}[0][1]$ with Hj=N-1 and Tj=0. Using these Hj and Tj values in Step 3, we get the following 3 CAM entries as shown in Figure 3.9(a).

CAM entry 2 and 3 can be combined into one new CAM entry $[1]^{Hi}[X]^{N-Hi}$. This is because CAM entries 2 and 3 differ only at bit 1 and 2: CAM entry 2 is [0][X], and CAM entry 3 is [1][0]. Since the bit 0 has to belong to the body, bit 1 and 2 cannot be [1][1] otherwise the bit has to be a header '0' after these contiguous '1''s. Therefore, [0][X] and [1][0] covers possibilities of the values in bit 2 and 3 and can be combined to [X][X]. This leads the combination of CAM entry 2 and 3 into a new $[1]^{Hi}[X]^{N-Hi}$.

The optimized algorithm requires a maximum of 2 CAM entries to map a range Wi to ∞ regardless of the code width N, as shown in Figure 3.9.

Entry 1 $[1]^{Hi-1}[0][X]^{N-Hi-Ti}[0]^{Ti}$ covers exactly all code words greater than or equal to Wi inside Wi's division, but excludes smaller numbers.

Entry 2 $[1]^{Hi}[X]^{N-Hi}$ covers all code words in divisions higher than Wi's by matching all or part of their header '1's, exploiting the growing nature of the header from division to division.

Wi $\quad [1]^{Hi-1}[0][0]^{N-Hi-Ti-1}[1][0]^{Ti}$

CAM Entry 1 $\quad [1]^{Hi-1}[0][X]^{N-Hi-Ti}[0]^{Ti}$

CAM Entry 2 $\quad [1]^{Hi}[X]^{N-Hi-3}[0][X]^{2}$
(Valid only when N-Hi>=3)

CAM Entry 3 $\quad [1]^{Hi}[X]^{N-Hi-3}[1][0]^{1}[X]^{1}$
(Valid only when N-Hi>=3)

Wj $\quad [1]^{N-2}[0][1]$

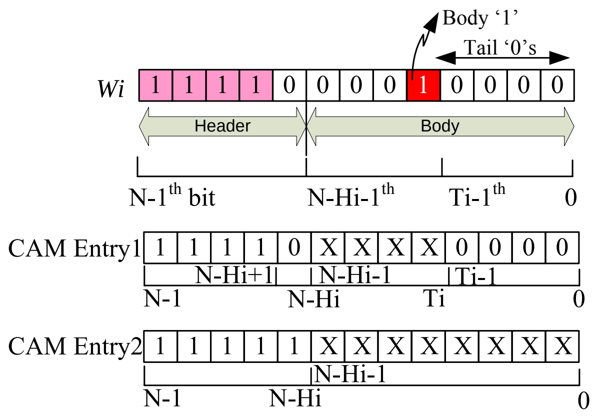
(a) The Pre-Optimized Three CAM Entries for the Range Wi to ∞

Wi $\quad [1]^{Hi-1}[0][0]^{N-Hi-Ti-1}[1][0]^{Ti}$

CAM Entry 1 $\quad [1]^{Hi-1}[0][X]^{N-Hi-Ti}[0]^{Ti}$

CAM Entry 2 $\quad [1]^{Hi}[X]^{N-Hi}$

(b) The Optimized Two CAM Entries for the Range Wi to ∞ in Regular Expression



(c) The Optimized Two CAM Entries for the Range Wi to ∞ in Diagram

Figure 3.9 An Optimize Algorithm to Map Range Wi to ∞ in Two CAM Entries

In the example shown in Figure 3.6, we will map the range of greater or equal than 24 into two CAM entries. The first CAM entry 110X,XXXX,XXX0, matches all code words with a header "110" and tail '0's at shortest "0", which covers code word 24 through 29 in division 2. The second CAM entry 111X,XXXX,XXXX will match everything in divisions 3 through 10, which includes numbers greater than 30. Therefore these two entries together cover everything from 24 to 64.

- **0 to Wj: “Less Than” A Specified Code Word**

The second special case is Wi=0 in the range Wi to Wj. The range 0 to Wj means less than or equal to a specified code word Wj.

When Wi=0, its code word is $[0]^{N-1}[1]$ with Hi=1 and Ti=0. Using these Hi and Ti values in Step 3, we get the following 3 CAM entries as shown in Figure 3.10(a).

CAM entry 1 covers the first division, where no header ‘1’ exists and only ‘1’ appears in the code word as a body ‘1’. It can be rewritten into 2 CAM entries as $[0][X]^{Hj-3}[0][X]^{N-Hj+1}$ and $[0][X]^{Hj-3}[1][0]^{N-Hj-Tj}[0]^{Tj+1}$, which are the two cases when bit N-Hj+1 is ‘0’ or ‘1’. The former entry can be combined with CAM entry 2 into one new entry $[X][X]^{Hj-3}[0][X]^{N-Hj+1}$. The latter entry can be rewritten as $[0][X]^{Hj-3}[1][0]^{N-Hj-Tj}[X]^{Tj+1}$ because code words in the first division won’t have another ‘1’ at bits 0 to Tj if bit N-Hj+1 is ‘1’. Thus, it can be combined with CAM entry 3 into a new entry $[X][X]^{Hj-3}[1][0]^{N-Hj-Tj}[X]^{Tj+1}$.

The optimized algorithm requires a maximum of two CAM entries to map a range 0 to Wj regardless of the code width N, as shown in Figure 3.10.

Entry 1 covers all numbers with a ‘0’ at its N-Hj+1th bit in divisions lower that Wj’s. It excludes all numbers in division Wj and any higher divisions, because in these divisions, the N-Hj+1th bit belongs to the header and must to be ‘1’.

Entry 2 $[X]^{Hj-3}[0][1][0]^{N-Hj+1}$ covers all numbers with a ‘1’ at its N-Hj+1th bit, in all divisions lower than Wj’s as well as in Wj’s division. These divisions have an equal or shorter header than Wj’s. It excludes numbers in divisions higher than Wj’s because of the ‘0’ in bit N-Hj. It also excludes the numbers greater than Wj inside Wj’s division because of the contiguous ‘0’s from bit Tj+1 to N-Hj.

In the example shown in Figure 6, we want to find the code words less than or equal to 60 (11000010). Entry 1 will be XXXX,XXX0,XXXX, which matches code words in division 0 except code word 4, code words in division 1 except code word 15, code words in division 2 except code word 25, code words in division 3 except code word 34, code words in division 4 except code word 42, code words in division 5 except code word 49, and code words in division 6 except code word 55. Entry 2 will be XXXX,XXX1,000X, which matches the code words in division 7, code word 60 in division 8, and code words 4, 15, 25, 34, 42, 49, 55. Thus these two CAM entries cover exactly all code words less than or equal to 60.

Wi $[0][0]^{N-2}[1]$

CAM Entry 1 $[0][X]^{N-1}$

CAM Entry 2 $[1][X]^{Hj-3}[0][X]^{N-Hj+1}$
(Valid only when Hj>=3)

CAM Entry 3 $[1][X]^{Hj-3}[1][0]^{N-Hj-Tj}[X]^{Tj+1}$
(Valid only when Hj>=3)

Wj $[1]^{Hj-1}[0][0]^{N-Hj-Tj-1}[1][0]^{Tj}$

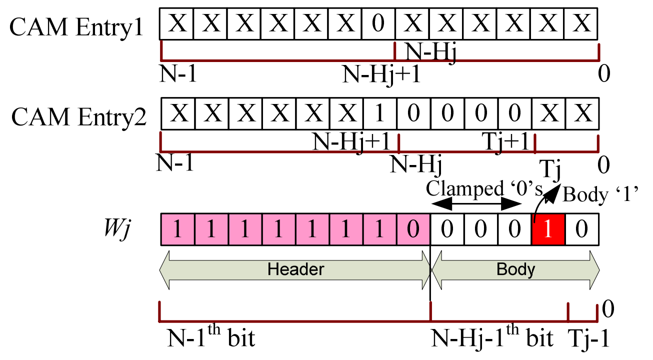
(a) The Pre-Optimized Three CAM Entries for the Range 0 to Wj

CAM Entry 1 $[X]^{Hj-2}[0][X]^{N-Hj+1}$
(Valid only when Hj>=2)

CAM Entry 3 $[X]^{Hj-2}[1][0]^{N-Hj-Tj}[X]^{Tj+1}$
(Valid only when Hj>=2)

Wj $[1]^{Hj-1}[0][0]^{N-Hj-Tj-1}[1][0]^{Tj}$

(b) The Optimized Two CAM Entries for the Range 0 to Wj in Regular Expression



(c) The Optimized Two CAM Entries for the Range 0 to Wj in Diagram

Figure 3.10 An Optimized Algorithm to Generate Two CAM Entries in Range Matching of "Less Than or Equal to Wj"

- **Within A Small Range Wi to Wj**

The third special case is a range between specified code words Wi and Wj when Wi and Wj lie within two divisions (Hj<Hi+2).

When Hj=Hi in which case Wi and Wj are inside the same division and have the same header, only 1 CAM entry is needed to map the range as shown in Figure 3.11. This entry maps to the common coverage of the CAM entry 1 and CAM entry 4 in Step 2. CAM entries 2 and 3 in Step 2 are invalid in the case because Hj-Hi-2 and Hj-Hi -3 both are less than 0.

Wi	$[1]^{Hi-1}[0][0]^{N-Hi-Ti-1}[1][0]^{Ti}$
CAM Entry 1	$[1]^{Hi-1}[0][0]^{N-Hi-Tj-1}[X]^{Tj-Ti+1}[0]^{Ti}$
Wj	$[1]^{Hj-1}[0][0]^{N-Hj-Tj-1}[1][0]^{Tj}$

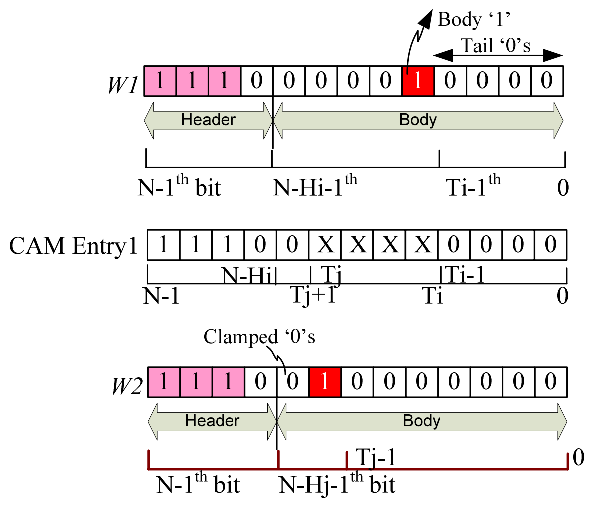


Figure 3.11 An Algorithm to Generate One CAM Entry in Range Matching of W1 to W2 Where W1 and W2 are in the Same Division

Wi	$[1]^{Hi-1}[0][0]^{N-Hi-Ti-1}[1][0]^{Ti}$
CAM Entry 1	$[1]^{Hi-1}[0][X]^{N-Hi-Ti}[0]^{Ti}$
CAM Entry 2	$[1]^{Hj-1}[0][0]^{N-Hj-Tj-1}[X]^{Tj+1}$
Wj	$[1]^{Hj-1}[0][0]^{N-Hj-Tj-1}[1][0]^{Tj}$

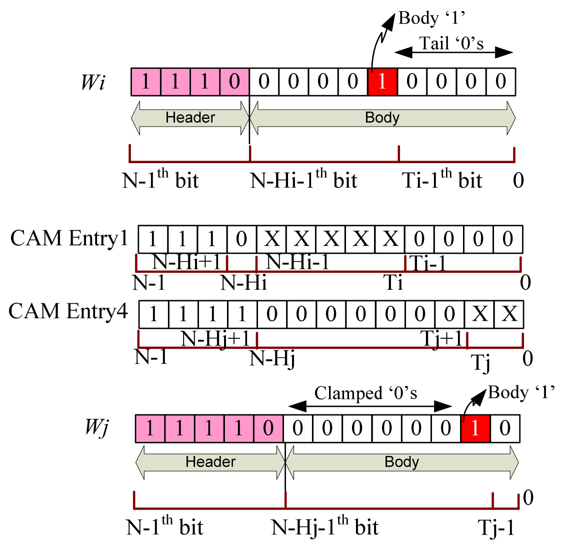


Figure 3.12 An Algorithm to Generate Two CAM Entries in Range Matching of (W1, W2) Where W1 and W2 are in Adjacent Divisions

CAM entry 1 $[1]^{Hi-1}[0][0]^{N-Hi-Tj-1}[X]^{Tj-Ti+1}[0]^{Ti}$ copies the header and tail'0's of Wi and clamped '0's of Wj. Other bits are filled with "X (don't care)". This covers all one-hot codes between Wi and Wj.

When Hj=Hi+1, Wi and Wj belong to divisions adjacent to each other and only 2 CAM entries are needed to map the range: the CAM entry 1 and CAM entry 4 in Step 2. CAM entries 2 and 3 are invalid in this case because Hj-Hi-2 and Hj-Hi -3 both are less than 0.

These two CAM entries are shown in Figure 3.12. CAM entry 1 $[1]^{Hi-1}[0][X]^{N-Hi-Ti}[0]^{Ti}$ covers numbers greater than or equal to Wi in Wi's division. CAM entry 2 $[1]^{Hj-1}[0][0]^{N-Hj-Tj-1}[X]^{Tj+1}$ covers numbers less than or equal to Wj in Wj's division.

3.4 SOFTWARE AND HARDWARE IMPLEMENTATION

Software implementation of a converter to convert an N bit binary code into the proposed hybrid code is shown in Figure 3.13.

```
for(code=0, i=N; i>0; i--)
{  if (number >i)
      {   number =number - i;
          code = code + (unsigned long)pow(2, (i+1));
      }
   else
         break;
}
code = code + (unsigned long)pow (2, number) +1;
```

Figure 3.13 Software Pseudo Code to Convert a N Bit Binary "Number" to the Corresponding Proposed Code Word "Code"

Hardware implementation of a counter of the proposed hybrid 4 bit code is shown in Figure 3.14. The architecture consists of replicated blocks including a 4 bit shift register and a control circuit with one register and combinatorial logic for one bit. This implements the function of a counter with the increment of 1 each time in the new code.

The hardware encoder and decoder can be implemented in a Look-Up Table (LUT), requiring one clock cycle to get the encoded or decoded result.

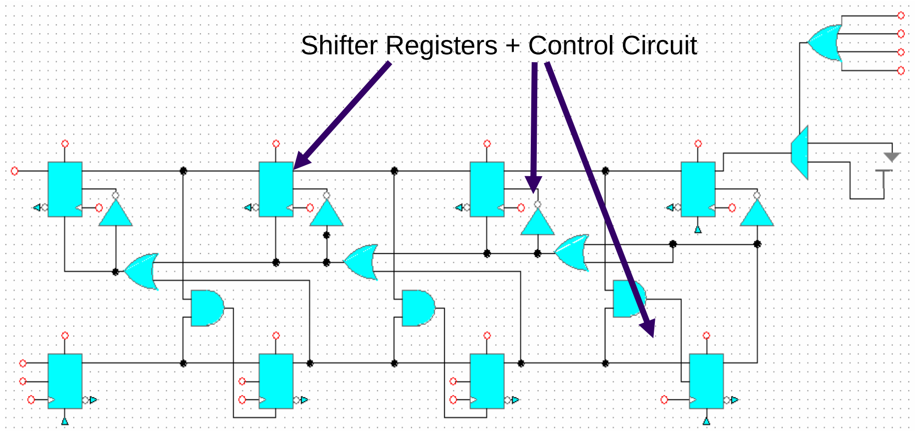


Figure 3.14 Hardware Implementation of a 4-bit Counter for the Hybrid Encoding Scheme

3.5 PERFORMANCE ANALYSIS AND COMPARISON

3.5.1 Performance Comparison

We compare the proposed hybrid encoding scheme with other existing codes listed in Section I, considering the depth scalability and the width scalability in a CAM.

Figure 3.15 compares the width expansions, which indicate the code density of the coding schemes considered. Assume the largest number these codes need to represent is *M*. Note that binary and Gray codes require the shortest data width of $\lceil \log_2 M \rceil$ bits, and that one-hot and Johnson encoding require the longest data width, *M* and *M/2* bits, respectively. In contrast, our proposed hybrid coding scheme is comparable to two-hot encoding whose width expansion is $\lceil \sqrt{2M} \rceil$.

Figure 3.16 shows the depth expansion required to map an arbitrary range. One-hot and Johnson encoding are the best choices, requiring only one or two CAM entries to map an arbitrary range, regardless of *M*. In contrast, binary code, gray code, and two-hot encoding increase the depth expansion as *M* increases. Even for small data values, the number of CAM entries required is significant. The proposed hybrid scheme is only slightly less efficient than

Table 3.1 Performance Comparison of Existing Encoding Schemes With the Hybrid Encoding Scheme

	Binary	**Gray**	**Two-hot**	**Proposed Hybrid Code**	**Johnson**	**One-hot**
Width Scalability: Data Width (bits) of the Code For Integer *M*	$\lceil \log_2 M \rceil$	$\lceil \log_2 M \rceil$	$\lceil \sqrt{2M} \rceil$	$\lceil \sqrt{2M} \rceil$	$\lceil M/2 \rceil$	M
Depth Scalability: Maximum Number of CAM Entries for an Arbitrary Range	$2\lceil \log_2 M \rceil$-2	$2\lceil \log_2 M \rceil$-3	$\lceil \sqrt{2M} \rceil$-2	3	2	1

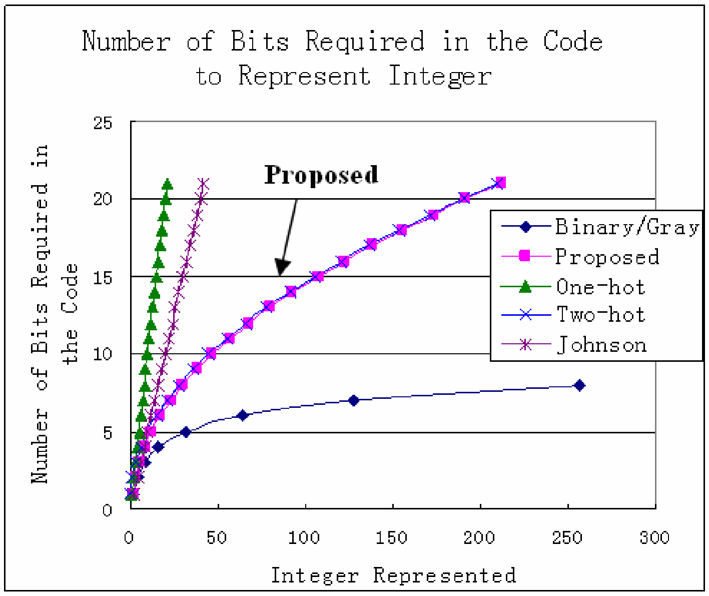


Figure 3.15 Comparison of Encoding Efficiency Among Different Encoding Schemes

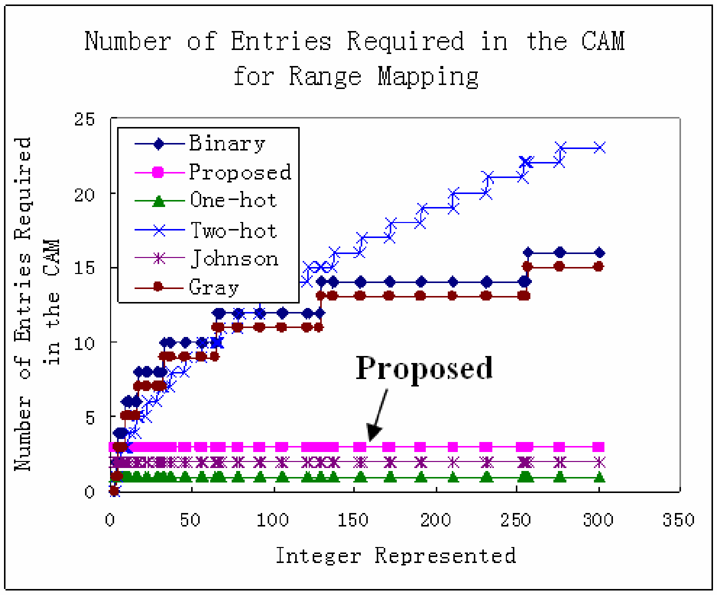


Figure 3.16 Comparison of Number of CAM Entries Required for an Arbitrary Range Matching in CAM Among Different Encoding Schemes

one-hot and Johnson encoding, requiring a maximum of three CAM entries for an arbitrary range regardless of M.

The performance comparison result is summarized in Table 3.1. Existing encoding schemes have scalability problems either in depth expansion or in width expansion for a CAM. Our proposed hybrid scheme represents an efficient tradeoff requiring only three CAM entries per arbitrary range and $\lceil \sqrt{2M} \rceil$ bits in width.

3.5.2 Performance in Real Applications

This unique hybrid encoding scheme has been applied to the payload offset range checking in a CAM-based Intrusion Detection System (IDS) called Snort. The maximum incoming packet payload length is 1500 bytes and it shifts 8 bytes at a time for string matching of IDS rules. The payload offset is incremented by 1 each time the payload shifts. Thus the proposed encoding scheme is used in the (0, 200) range. The Snort IDS will check if the current payload offset fits the range matching in any of the thousands IDS rule in parallel. In this application of range matching, only 20 bits of the 576 bit CAM are required to represent payload offset, leaving enough bits in width for other criteria checking different fields in the packet, such as IP address or port number. In this application, the proposed coding scheme achieves the CAM depth scalability and width scalability substantially better than any of the other schemes considered.

3.6 CONCLUSION

In this chapter, we presented a unique hybrid encoding scheme that fits efficiently into a CAM to perform range matching with a normal search operation. The depth expansion in a CAM using this hybrid code is deterministic: up to 3 entries for matching any arbitrary range without regard to key width. The width expansion using this hybrid code grows as a factor of the square root of the number of bits in the word. This unique hybrid encoding scheme is compatible with the size of commercially available CAMs and performs the range matching more efficiently than existing encoding schemes. Algorithms to generate the set of new code words, encoding and decoding of the code, and generate CAM entries for arbitrary range matching are provided in this paper. This unique encoding scheme has been applied to payload offset checking in a CAM-based Intrusion Detection System (IDS) Snort.

The depth expansion in a CAM using the proposed encoding scheme is deterministic: up to 3 entries for any arbitrary range regardless of the width of the keys. The width expansion grows as a square root of the number need to be encoded. This unique encoding scheme reduces the size of the CAM and performs the range matching more efficiently than existing encoding schemes.

Chapter 4 and Chapter 5 utilize this hybrid range matching scheme for single-packet and multi-packet attack detection.

4.0 DETECTION OF SINGLE-PACKET ATTACK

The single-packet attack is the simplest and most widely used attack form on the Internet. Twenty-four attack types of the 58 network intrusions in the Lincoln Lab intrusion detection evaluation dataset are single-packet attacks [68]. Rules in an open source IDS Snort [81, 82] are 100% single-packet attacks detection. In single-packet attack detection, packet information is examined to find attack signatures inside the packet. It involves packet header checking and packet payload string matching. Detection of single-packet attack is the basic Intrusion Detection function and is the foundation of other complicated network intrusion detection including multi-packet attacks.

Snort is an open-source, lightweight network intrusion detection system (NIDS) that logs and analyzes traffic on IP networks in real-time [81, 82]. It is a single-packet attack detection method, which means only the current packet information is checked to find attacks. No previous packet information is used. This feature classifies the Snort rules into our single–packet detection rule. Therefore, we use Snort as an example in our single-packet detection research. As software-based IDS, Snort has serious performance limitations and scalability problem when new rules are added.

Content Addressable Memory (CAM) [77] is a parallel search engine which searches its memory content in parallel to find one or more equals to the input data. It has been proposed to accelerate many network functions such as packet classification and intrusion detection that are search-intensive by nature. Compared to software-based network function approaches, ternary CAMs offer many benefits. Unlike software-based solutions that perform serial searches and cost linear search time, CAM performs a search operation in parallel and takes a constant number of clock cycles, typically tens of nanoseconds in current technology. This is 100 times to 1000 times faster than software running on processors. In addition, the search time is independent of the number of rules used in a given search. This consistently high performance motivates its application in single-packet IDS.

In order to map the Snort rules into a CAM for high performance parallel search, we analyze the rule requirements of packet checking in all Snort rules. From version 2.0 to version 2.4, the total number of Snort Rules (unregistered user's version) grew from 1977 to 3191. All the rule requirements in these rule sets are listed as follows:

- Header checking (100% of rules):
    1. Scalar Matching in Headers (96.4% - 97.7% of Snort rules)
    2. Range Matching in Headers (2.3% - 3.6% of Snort Rules)
    3. Two Fields Comparison in Headers (0.03% - 0.05%)
- Payload string matching (92.7% - 95.3% of rules)
    4. Single Short String Matching in CAM (34.0% - 55.2%)
    5. String Matching with Payload Offset range Criteria (16.7% - 32.0%)
    6. Matching of Multiple Independent Strings (7.0% - 10.0%)
    7. Matching of Long Strings (>26 Bytes) (3.3% -5.3%)
    8. Distance Range Matching Between Multiple Strings (12.0% - 27.5%)
    9. Variable String Matching with Perl Regular Expression, or *threshold*, *byte_jump* and *byte_test* keywords (8.6% - 43.0%)

From the rule analysis we found that in version 2.0 through version 2.4, 39.0% - 63.0% of rules only consist of requirements 1 through 4, which are easy to implement in hardware. Another 17.9% -28.4% of the rules are solved when including requirements 5 through 8. These rules are more complicated to implement in hardware and are increasing in number among all rules. These complicated requirements are becoming an important part of the rules in the current Snort version 2.4. 8.6% - 43.0% of the rules involving requirement 9 which requires string matching of variable length and are presented as a Perl Regular Expression. This type of matching and is very difficult to map to a CAM whereas they can handle fixed length string matching function directly. This type of rules has increased in number drastically from version 2.0 to version 2.4.

Previous work [83] has provided solutions for requirements 1, 2, 3, 4 and partial solutions for the more complicated requirements 5, 6, 7, 8. The partial solution basically solves only requirement 4, and then use an external processor to process the extra information in requirements 5, 6, 7, 8. However, the growth in complicated requirements motivates dedicated hardware solutions for requirements 5-8. Requirement 9 would be easier to be process by a processor instead of a CAM.

In this chapter, we introduce new processing methods using CAM to perform complicated requirements 5, 6, 7, and 8, based on the novel range mapping encoding scheme presented in Chapter 3. This solves extra 17.9% -28.4% of Snort rules besides the previous solved 34.0% - 55.2% of rules. A prototype architecture has been developed to handle 91% of the Snort rules (version 2.0) and all their requirements, while the remained 9% of rules contained requirement 9.

The structure of this chapter is as follows: in Section 4.1, we introduce the basics of Snort and its rules. A performance analysis is discussed which shows the very limited performance Snort software has for faster networks. In Section 4.2, previous work of Snort mapping to CAM is reviewed. We discuss Repanshek's CAM-Assisted Snort solution [83] in more depth since it is the most related and detailed document to this research. In Section 4.3 through 4.6, novel solutions that only use CAM plus RAM are introduced for requirement 5, 6, 7, 8, respectively. Section 4.7 presents the prototype system that has been developed for this architecture, as well as the mapping result for 91% of the Snort rules (version 2.0) to this architecture.

4.1 INTRODUCTION OF THE OPEN SOURCE SNORT IDS

Snort is an open-source, lightweight network intrusion detection system (NIDS) that logs and analyzes traffic on IP networks in real-time [81, 82]. It analyzes ongoing traffic and detects various types of attacks through pattern matching techniques against "signatures" (also called rules) that describe the patterns of known attacks [81, 82]. Because of its open source and popularity, we use Snort as an example in our single- packet detection research.

4.1.1 Snort Rules

The source code of Snort version 1.9.0 consists of several important software components: snort.c (2600+ lines), parser.c (4100+ lines), decode.c (3800+ lines), detect.c (1900+ lines) and plug-ins files (48 files) [84]. A block diagram of their roles and interactions is shown in Figure 4.1.

Each captured packet is processed by functional subsystems in the following order: Packet Decoder, Preprocessors, Detection Engine, and Output Systems. The detection engine and Snort rules are at the core of the intrusion detection process. Snort rules (also called signatures), describe the characteristics of intrusions and are parsed at the beginning of the detection process. They are stored into a three-dimensional Linked-List by the parser.c program. The detection engine in Detect.c detects intrusions by searching the three dimensional Linked-List for a match. More details are explained in the following sections.

The format of a Snort rule has two parts: the rule header and the rule option [81, 86], as shown in Figure 4.2.

A rule header contains six parts [81, 86]: Rule Action, Protocol, Source IP, Source Port Number, Destination IP and Destination Port Number.

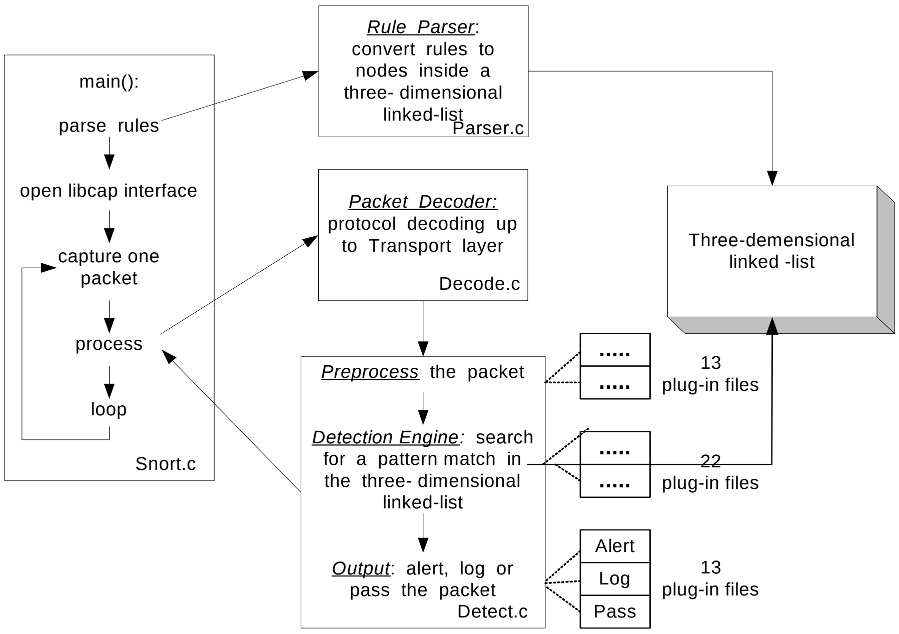

main():
parse rules
open libcap interface
capture one packet
process
loop
Snort.c
Rule Parser: convert rules to nodes inside a three- dimensional linked-list
Parser.c
Packet Decoder: protocol decoding up to Transport layer
Decode.c
Three-demensional linked -list
Preprocess the packet
Detection Engine: search for a pattern match in the three- dimensional linked-list
Output: alert, log or pass the packet
Detect.c
.....
.....
13 plug-in files
.....
.....
22 plug-in files
Alert
Log
Pass
13 plug-in files

Figure 4.1 Block Diagram of the Snort Source Code (Version 1.9.0)

Rule Header	Rule Option
[Rule Acrion] [Protocol] [Source IP] [Source Port] <-> [Destination IP] [Destination Port]	([Option Keyword1] : " [****] " ; [Option Keyword2] : " [****] " ; [Option Keyword3] : " [****] " ; [Option Keyword4] : " [****] " ; [Option Keyword5] : " [****] " ; ....)

Figure 4.2 Format of a Snort Rule

- Rule Action includes the options “Activate”, “Dynamic”, “Alert”, “Log” and “Pass”, which indicates the action to be taken if this rule is matched with a packet. “Activate” means to alert the system and turn on a corresponding “Dynamic” rule. “Dynamic” means that the rule remains idle until activated by an “Activate” rule and it would then act like a “log” rule. “Alert” means to alert the system with a message while the packet information is recorded into the file. “Log” means to record the packet to a file. “Pass” means let the packet go without any recording or alerting.
- Protocol indicates which protocol this rule is monitoring, such as IP, TCP, UDP, ICMP, and ARP.
- Source IP and Destination IP fields are described using the Classless Inter-Domain Routing (CIDR) format [87] that includes a numeric IP address with a slash and a network mask.
- Source port number and destination port number are shown as straight numbers.
- Rule Options are shown within a bracket after the rule header to flexibly describe more characteristics of the rule. A single rule can have many rule options separated by semicolons. Each option starts with the Option keyword and a “:”. Content for these options are enclosed inside quotation mark after “:” Example option keywords include the following, among others:

  “msg” to indicate the text to print when an alert happens,

  “ttl” to test the IP header’s time-to-live field value,

  “content” to search for a pattern in the payload of the packet, and

  “nocase” to indicates string matching is case-insensitive.

  A complete list of available keywords can be found in the Snort Manual [10].

  An example of Snort rules is listed as follows:

alert tcp any any -> 192.168.1.0/24 21 (content: "USER root"; nocase; msg: "FTP root user access attempt";)

This rule means that the system will be sent with an alert with the message "FTP root user access attempt" when the incoming packet complies with the following four conditions:

- It is a TCP packet.
- It comes from any IP address, any port number.
- It targets the IP address 192.168.1.0 with a sub-net mask 24, which means a class C network with IP addresses ranging from 192.168.1.0 to 192.168.1.255, and to port number 21.
- It contains case-insensitive strings "USER root" in the payload.
- The alert action will log the alert message "FTP root user access attempt", together with the full packet header, into an ASCII file.

4.1.2 Snort's Three-Dimensional Linked-List and Detection Engine

All Snort rules are parsed and stored into a three-dimensional Linked-List before packet capturing and detection begins [88]. The structure of the three-dimensional Linked-List is shown in Figure 4.3. The three-dimensional Linked-List actually consists of five action lists (which are underlined in the left part of Figure 4.3) according to the different rule action keywords: Activate, Dynamic, Alert, Pass and Log which were explained in Section 2.4.1.

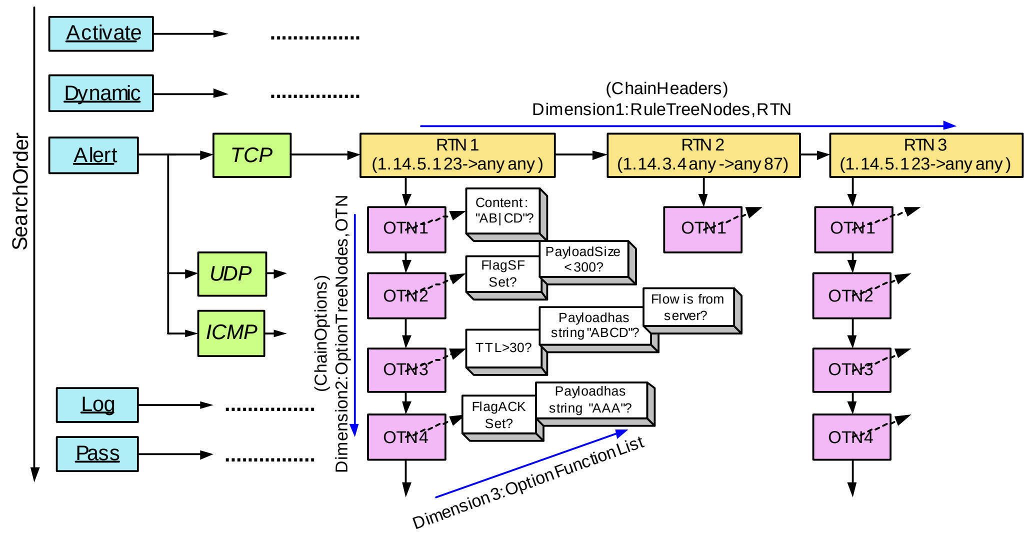

SearchOrder
Activate
Dynamic
Alert
Log
Pass
TCP
UDP
ICMP
(ChainHeaders)
Dimension1:RuleTreeNodes,RTN
RTN 1 (1.14.5.123->any any)
RTN 2 (1.14.3.4 any->any 87)
RTN 3 (1.14.5.123->any any)
(ChainOptions)
Dimension2:OptionTreeNodes,OTN
OTN1
OTN2
OTN3
OTN4
Content: "AB|CD"?
PayloadSize <300?
FlagSF Set?
Flow is from server?
Payload has string "ABCD"?
TTL>30?
FlagACK Set?
Payload has string "AAA"?
Dimension3:OptionFunctionList

Figure 4.3 Structure of the Snort Three-Dimensional Linked-List (Adapted from [12])

If we look into these five action lists, each consists of three protocol sub-lists according to the different network protocols e.g. TCP, UDP and ICMP that they monitor.

If we delve deeper into each of these protocol sub-lists, an independent three-dimensional Linked-List appears. As we can see in the three-dimensional Linked-List, dimension 1 consists of many Rule Tree Nodes (RTNs), also called chain headers [81, 86]. Each RTN records a data pair including source destination IP and port number. Dimension 2 consists of many Option Tree Nodes (OTNs), also called chain options [81, 86] (marked as OTN in Figure 4.3). Each OTN records all rule options for a single rule. For a Snort rule, the rule header is mapped into an RTN in Dimension 1, while the rule options are mapped into a single OTN in Dimension 2 under that RTN [81, 86]. Thus, a single rule would be mapped in a single OTN somewhere inside the three-dimensional Linked-List. Different rules with identical values in rule headers are mapped into the same RTN, but different OTNs.

There is a virtual Dimension 3 which contains a list of comparison functions. Each RTN and OTN saves a pointer for its own list of comparison functions that need to be executed to determine if there is a match between itself and a processing packet. Therefore the overall three-dimensional Linked-List actually contains many small-scale three-dimensional Linked-List categorized into different action lists, and protocol sub-lists.

The source code of the detection engine appears inside Detect.c, which implements the critical searching task in Snort. The detection engine searches recursively in the three-dimensional Linked-List for a pattern match on any incoming packets [88]. It searches in each action list in the order shown in Figure 4.3, from “Activate” list to “Pass” list, as well as in each protocol list. Inside the Linked-List under a protocol list, the detection engine first searches RTNs in Dimension 1 to see if there is one that complies with the current processing packet. This is done by calling comparison functions in the Dimension 3 of each RTN. When an RTN has all functions in its Dimension 3 return “true” values, it means this RTN matches the processing packet. In this case, the detection engine will proceed down in Dimension 2 from that matched RTN and call all functions in the Dimension 3 for each OTN to search for a match. Return values being “true” for all functions indicate a match between the complete rule and the packet being processed. A match between the current processing packet and a specific rule will cause the action described in the rule to be taken. A match will also stop the search. No match will cause the detection engine continue to search the rest RTNs and OTNs until all possible matched nodes in the three-dimensional Linked-List have been evaluated.

Among all functions in Dimension 3, the string matching function caused by option keywords “Content” and “Uricontent” is the most time-consuming as shown by statistics shown later in this chapter. Therefore, the string-matching function is placed at the end of the function list in order to avoid unnecessary operations. Currently, Snort uses the Boyer-Moore (B-M) algorithm to find a string inside the payload of a packet [81, 89]. This B-M algorithm is good at searching a sub-string among multiple strings, but it doesn’t take advantage of similarities among different Snort rules [89]. This results in a large performance bottleneck.

4.1.3 Performance Analysis of Snort Using Code Profiling GProf

GNU Profiler (Gprof) is a code profiler program that calculates the execution time for a program and for each function inside the program [90]. In order to get the time profile for an evaluating program, the user needs to compile and link the program with a profiling enabled option before executing the program [90]. The profile can be analyzed by GProf to get a text file that shows the execution time for the program and for each function inside the program [90].

GProf software is installed on our Linux box and will be used here to analyze the performance of the Snort version 1.9.0. The following experiments were conducted with Snort version 1.9.0 running on a Dell Power-Edge 4400 server with dual 866MHz Pentium III Xeon processors, 1GB RAM and running the Redhat 7.2 OS. Six different network traffic clips were sampled in a medium-sized Electrical Engineering research lab and saved in binary tcpdump format “.log” file. Three types of performance are analyzed.

First is the performance analysis of Snort detection rate. As shown in Figure 4.4, the detection rate ranges from 30-300Mpbs with 1227 Snort rules in the experiments, according to different average size of packet files. The detection rate goes up as the average frame size increases. This is because once the larger packet’s header doesn’t match rule’s header criteria, the whole packet is skipped without checking its long payload. Therefore the processing rate goes up. In normal network, 200 -300 bytes long packet is the most common. In that range, the detection rate is only around 50Mbps, much less than the current Gigabit network requirements.

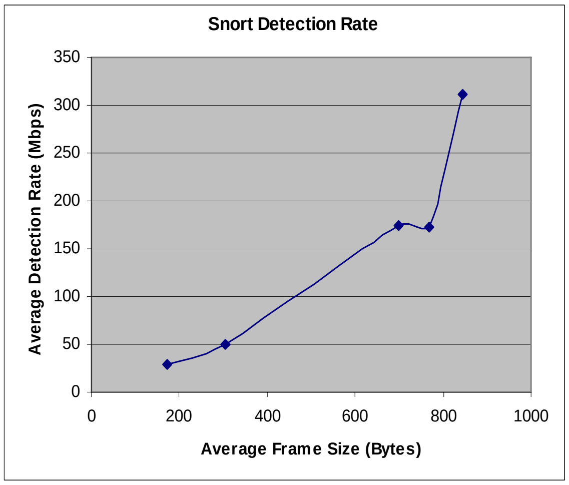


Figure 4.4 Snort Detection Rate 30-300Mbps for Various Packet Sizes

Second is the performance analysis of the time spent on different functional blocks in Snort. The Detection Engine consumes most of the time among the four functional blocks. Inside the Detection Engine, five major categories of functions are defined and shown in Table 4.1. Their processing time distribution inside the Detection Engine is presented in Figure 4.5, and their execution time expressed as a percentage of total Snort time is shown in Table 4.2.

It is apparent from Figure 4.5 that four categories inside the Detection Engine require the most significant amount of time. First are the String Matching functions that include mSearch and mSearchCI that perform string matching using the Boyer-Moore algorithm [81, 89], with case sensitivity and case insensitivity respectively. They always ranked first in our analysis of Packets 1-6, and cost 25%-35% of total time as shown in Table 4.2. Second are the 3D Linked-List functions, which include functions EvalHeader and EvalOpts to go though Dimension 1 and Dimension 2 respectively, and call different comparison functions attached with the tree node in Dimension 3. They always ranked second in our experiment and cost 20%-25% of total time. Third is Pattern Match Overhead which includes functions CheckUriPatternMatch, CheckANDPatternMatch, uniSearch and uniSearchCI, which calculate and set up the parameters for exact string matching. Pattern Match Overhead always ranked third in our experiments and cost 10%-20% of total time. The last category includes Packet Header Checking functions which check the header fields against rules. It includes the functions CheckDstPortEqual, CheckSrcIP, CheckIpOptions and other header checking functions. Together these functions cost around 7%-15% of total time. The remaining 300 functions in the Detection Engine cost less than 8% of total time. Therefore, the top two categories String Matching and 3D Linked-List are serious bottlenecks which together cost about 50% of total time.

Table 4.1 Five Major Categories of Functions in Snort Detection Engine

Detection Engine	
Categories	**Function Name**
String Matching	mSearch, mSearchCI
3D Linked-List	EvalOpts, EvalHeader
Pattern Match Overhead	CheckUriPatternMatch, CheckANDPatternMatch, UniSearch, UniSearchCI
Packet Header Checking	CheckDstPortEqual, CheckSrcIP, CheckIpOptions, etc.
Others	The Remaining 300 functions

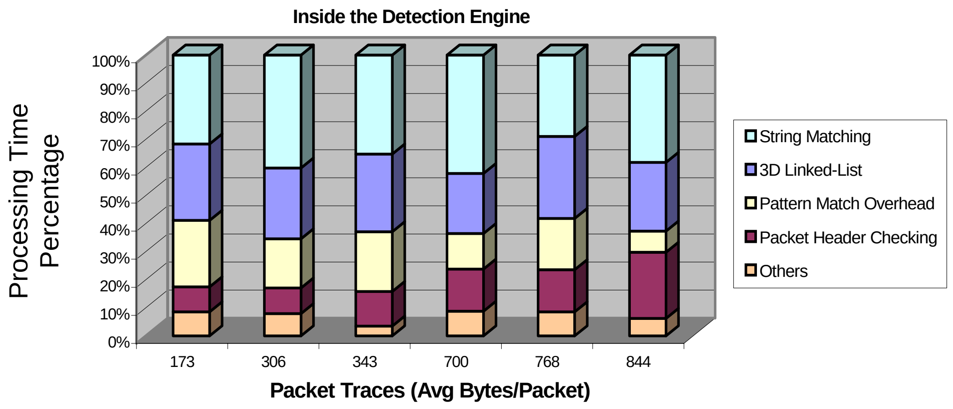


Figure 4.5 Execution Time Distribution Inside Snort Detection Engine

Table 4.2 Execution Time Percentage for Functions in Snort Detection Engine and Preprocessor*

Note : Percentage Shown in Table is the percentage of total Snort execution time

Functions		**Packet Traces**						Appr. Range
		1	**2**	**3**	**4**	**5**	**6**	
Detection Engine	String Matching	23.71%	33.26%	32.16%	35.54%	24.11%	31.30%	25%-35%
	3D Linked-List	21.13%	21.08%	25.06%	18.15%	24.57%	20.07%	20%-25%
	Pattern Match Overhead	17.88%	14.92%	19.23%	10.5%	15.73%	6.20%	6%-20%
	Packet Header Checking	6.78%	7.64%	11.12%	12.69%	12.53%	18.78%	7%-15%
	Others	6.58%	6.56%	3.09%	7.35%	7.19%	4.94%	3%-8%
Preprocessors		19.09%	12.50%	4.65%	12.42%	10.95%	12.50%	5%-20%
Others		4.83%	4.04%	4.69%	3.35%	4.92%	6.21%	3%-6%

The third performance analysis is the effect of different number of rules on Snort execution time. Figure 4.6 shows the effect of different numbers of rules applied in Snort towards the execution time of Snort. The total execution time includes time for initializations, protocol decoding and intrusion detection. From Figure 4.6, we can see that the total Snort offline execution time and the exact intrusion detection time (which is the time spent in the function "Preprocess" and its "child" functions) both rise linearly when the applied number of rules increases. We can also find that the rise of intrusion detection time is the main reason for the increase in total execution time. As the number of the rules (and perhaps the complexity of the rules) expands to detect future attacks, the execution time per packet is expected to rise at least linearly and thus the Snort throughput will decrease linearly or worse.

Table 4.3 Different Sizes of Rule Sets and 3D Linked-List in Snort

Rule Set No.	# of Rules	# of Rule Tree Nodes (RTNs)	# of Option Tree Nodes (OTNs)
1	0	0	0
2	1	1	1
3	10	6	10
4	100	31	100
5	401	40	410
6	659	101	659
7	899	107	899
8	1120	124	1120
9	1273 (default)	133	1273
10	1500	178	1500
11	1700 (all)	192	1700

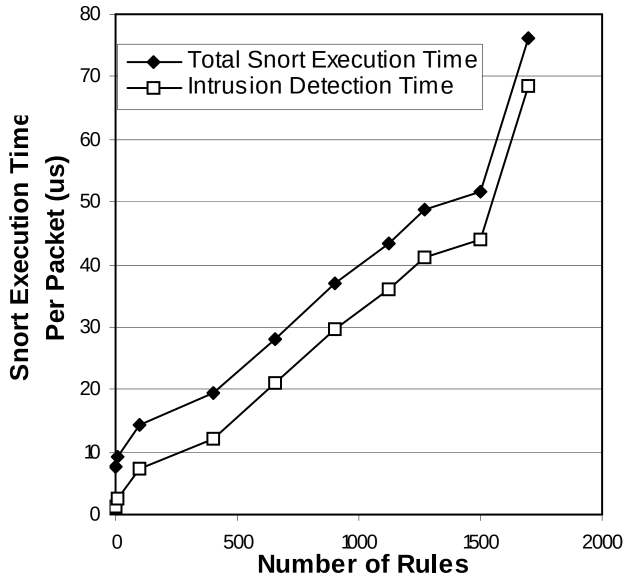


Figure 4.6 Snort Execution Time When Number of Rules Increase

4.1.4 Growing Number of Rules among Various Versions

It is predictable that the number of rules is growing as new versions of Snort replace older versions. The additional rules describe more network attacks and thus help to detect a more complete set of intrusions existing in current networks. As a matter of fact, not only is the number of rules growing, but the rule requirements are also becoming more complicated. Figure 4.7 shows the Snort total rule number growth (the height of the 3 division bar), as well as the growth of the number of complicated mapping rules and hard to map rules. From the figure, we can see a total rule number growth of 25% between version 2.0 and version 2.1, as well as a stable 8.2% -9.6% growth from version 2.1 up to version 2.4. The number of simple mapping rules which only involves rule checking requirements 1-4 remains similar. The number of complicated mapping rules which involves rule checking requirements 5-8 grows slightly. The number of hard to map rules which involves rule checking requirement 9 increases drastically.

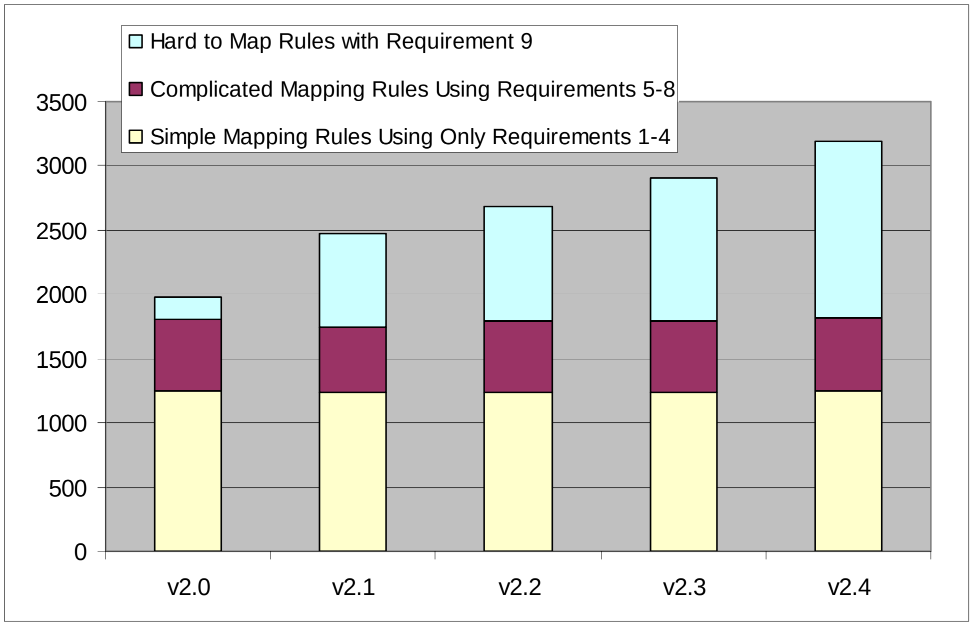


Figure 4.7 The Growing Number of Rules and Rule Checking Reqirements for Snort Version 2.0 - 2.4

Figure 4.8 shows the analysis of detailed rule checking requirements mentioned previously from version 2.0 through 2.4. In the figure, the dashed red line represents simple rule checking requirements 1-4. The green solid line represents complicated rule checking requirements 5-8. The solid black line represents hard to map rules with requirement 9.

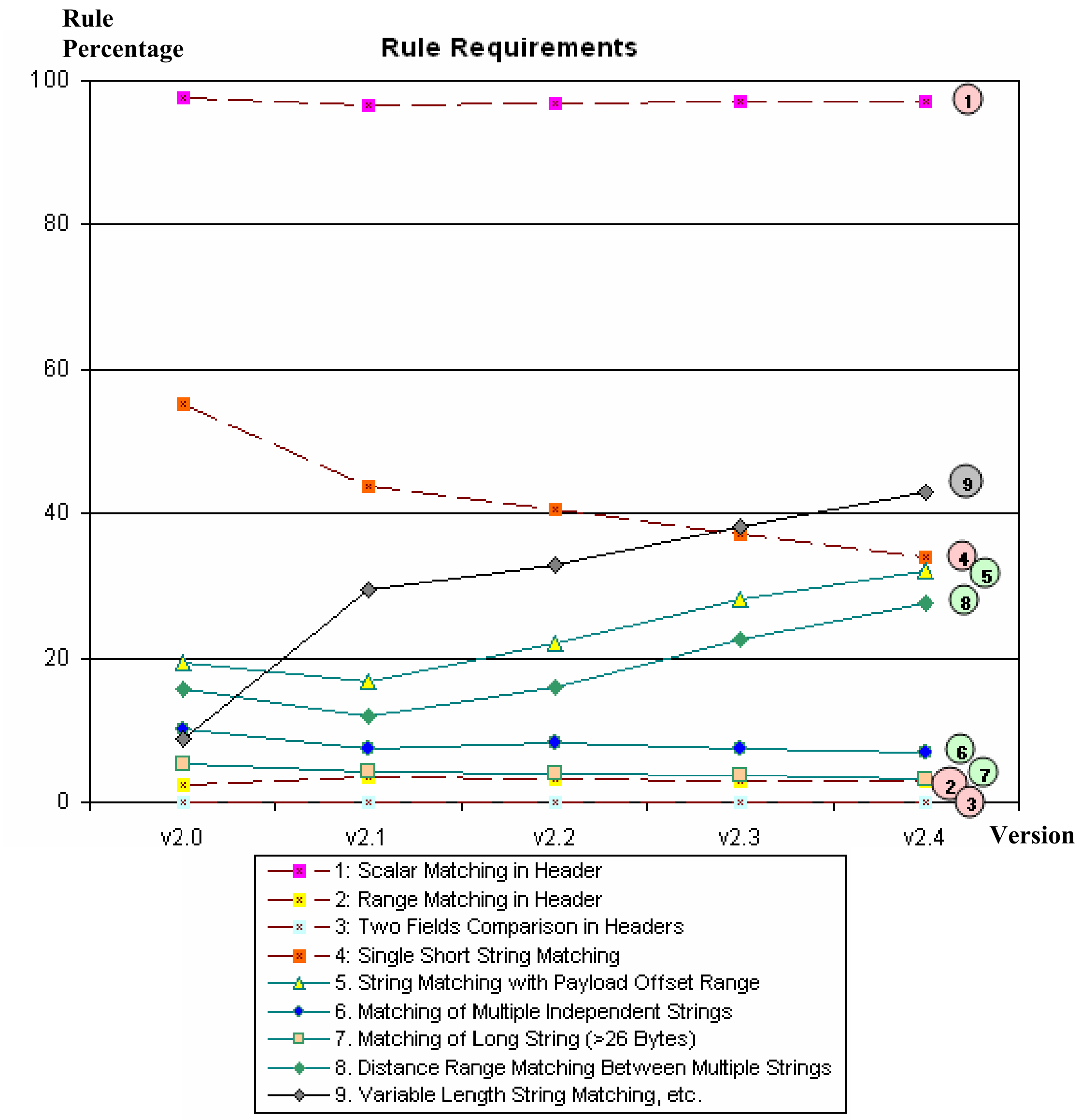


Figure 4.8 Detailed Rule Checking Requirements of Snort Version 2.0 through Version 2.4

From the figure, some interesting facts are observed. First, requirement 1 of scalar matching remains dominant among all the rules. Second, simple requirements 2, 3 and 4 either stay very at a very small percentage or decline significantly. In contrast, the complicated requirements 5, and 8 grow considerably in percentage. Requirements 6 and 7 grow in number (not shown in the figure) although the percentage declines slightly. Hard to map rules with requirements 9 increases greatly. It is also noticeable that from version 2.0 to version 2.1 the percentage curves for requirements 5 and 8 decline and don't quite match the increasing curve from version 2.1 to 2.4. This is because between version 2.0 and later versions there is a major change of rule style. Version 2.0 uses none of the "Perl regular expression" but the later versions use more and more of it. In version 2.4 more than 33% of rules have Perl regular expression criteria. When rules are added for versions 2.1 through 2.4 of Snort, a big percentage (50%-100%) of the addition is requirement 9 rules. Therefore, though the requirements 5, 6, 7 and 8 all grow in number from version 2.0 to 2.1, their percentages decline because the total number of rule has been raised significantly by requirement 9. This explains the decline trends in requirements 5 and 8 from version 2.0 to 2.1, as well as the decline trends in requirements 6 and 7 from version 2.0 to 2.4.

In conclusion, both the number of rules and the percentage of complicated rule requirements increase from Snort versions 2.0 to 2.4. These complicated rule requirements play a more and more important role in Snort. The growth of total rule number motivates CAM-based parallel solution for Snort to avoid further performance deterioration due to more rules. The growth of complicated rule checking requirements motivates development of a dedicated CAM-based solution for it instead of the current partial solution.

4.2 PRIOR WORK

There are several attempts to use hardware to accelerate Snort IDS. We review previous work of Snort mapping to FPGA and CAM in this section. In addition, Repanshek's CAM-Assisted Snort solution [83] will be discussed in more depth here since it is the most related and detailed document to our research.

4.2.1 Prior work of Snort Implementation in Hardware

The method proposed in [91] proposes to map the Snort intrusion detection engine in reconfigurable hardware on an FPGA. An early version of Snort (v.1.8.7) that supported 1239 rules is used in this research. In this architecture, a header checking is performed first to check source and destination addresses and ports, as well as protocol type. Then a payload string matching is performed if the header search returns any match. This research suggests the use of CAMs for header and payload searches. While it proposes many of the basic concepts for Snort to be mapped in hardware, there are no design details or results available to support a proof of concept. In addition, the choice of an FPGA as a target for the architecture is impractical.

The architecture proposed in [92] is another attempt to map Snort rules directly into reconfigurable hardware. The design features rule-specific content pattern-matching engines that match payload chunks in 4 byte increments against a given rule. Each rule, represented by its own content pattern-match engine described in VHDL, is searched in parallel. First, packet header checking is performed. If there is a match, 4 byte shifts of payload information are fed into the rule's individual content pattern-matching engine to find any match. Only 105 Snort rules are implemented in this design, and no details is given how these rules are mapped to VHDL. This architecture relies on frequent reconfiguration to account for new rules, hence its FPGA target.

The Granidt (Gigabit Rate Network Intrusion Detection Technology) architecture is proposed in [93]. This work proposes an integrated hardware/software solution to improve Snort performance with CAMs. The software "rule compiler" generates a hardware representation of the rule fields to be matched. The "rule processor" software component initializes the hardware CAMs and initiates packet processing. Header and payload searches are performed separately, by a number of individual CAMs. A match vector that indicates the results of the CAM searches is correlated to determine the appropriate action for a matched rule. Since the rule compiler is in software, new rules can be added to the design without resynthesis. This design requires several individual CAMs, which is cost-prohibitive. Also, the supported signature length is limited to 20 bytes, which is not able to support all Snort rules. Details of the mapping of the rules themselves into hardware are not clearly defined.

4.2.2 Repanshek's CAM-Assisted Snort Solution [83]

Repanshek [83] has provided solutions for the requirements 1, 2, 3, 4. He also provided a partial solution for requirements 5, 6, 7, and 8.

Solution for Scalar Matching in Headers

For header checking, a direct CAM mapping is used for header scalar matching that checks the values of some specific fields of the packet header. Repanshak assembles these fields into a single vector, and then matches this vector from the incoming packet with the CAM criteria vectors in its CAM entries.

A rule in CAM can be described by the value of each bit in the CAM. In order to implement the ternary value,'1','0', and 'X', two bits are need for one value. The first bit is called the "Data bit" which indicate if the CAM bit is '1' or '0'. The other bit is called the "Mask bit" which indicates if this bit is an 'X' or not. If the "Mask bit" is 1, that means the "Data bit" is valid and it is the value of this ternary CAM bit. If the "Mask bit" is 0, that means the "Data bit" is invalid and this ternary CAM bit is "X". A rule can put the value into the fields which have a specific value in the rule, and "don't care" bits in the fields with no value specified in the rule.

Solution for Range Matching in Headers

Another kind of header checking involves range matching that checks whether the values of some specific fields of the packet header lie within a certain range. Snort includes range matching in port number field, payload size, TTL fields and IP address fields. Repanshak uses one-hot encoding to represent all ranges in Snort header. Extra comparators before the CAM are required to pre-process the packet. Also extra bits are required in the CAM to indicate the range comparison result.

Solution for Two Fields Comparison in Headers

Another header checking involves two field comparison that compares the values of two fields in the headers. For example, it could test whether the Source IP address equals to the Destination IP address. This is very difficult to be map directly into CAM. Repanshak proposes the use of an extra comparator before CAM to do the comparison and the use of one extra bit in CAM to indicate if these two fields are the same or not.

Solution for Single Short String Matching

Payload string matching is a process to determine if a substring (also called "string matching key") is included in a long string (in IDS domain, it is packet payload). It is a more complicated process than header checking, where all fields are at fixed positions. The string matching key can appear at any position inside the payload. This leads to its computational intensive nature. String matching has been the most serious performance bottleneck in rule-based IDSes [23, 24], and cost around 1/3 of the total execution time in Snort IDS in our profiling experiments. In addition, the execution time of string matching process will increases linearly as the number of signatures rises when new signatures are added to identify future new attacks [23].

Repanshek uses a brute-force matching in CAM that can solve single string matching per rule with the key string shorter than the CAM width allocated for string matching.

The details of implementing string matching function in CAM are shown as follows: Packet payloads are shifted one byte at a time (one ASCII character) and compared with the string in CAM. If the payload does contain a string at K^{th} bit, at time K, it will shift to the beginning and be matched with the CAM content.

In order to improve the string matching speed, more string keys can be copied with shifted position in CAM. Therefore, the payload string can be shifted more than one byte at a time. In the example shown in Figure 4.9, we duplicated the string key in the CAM, and the payload string can be shifted 2 bytes at a time which doubles the string matching processing rate. Therefore, CAM space can be traded for processing rate in the design.

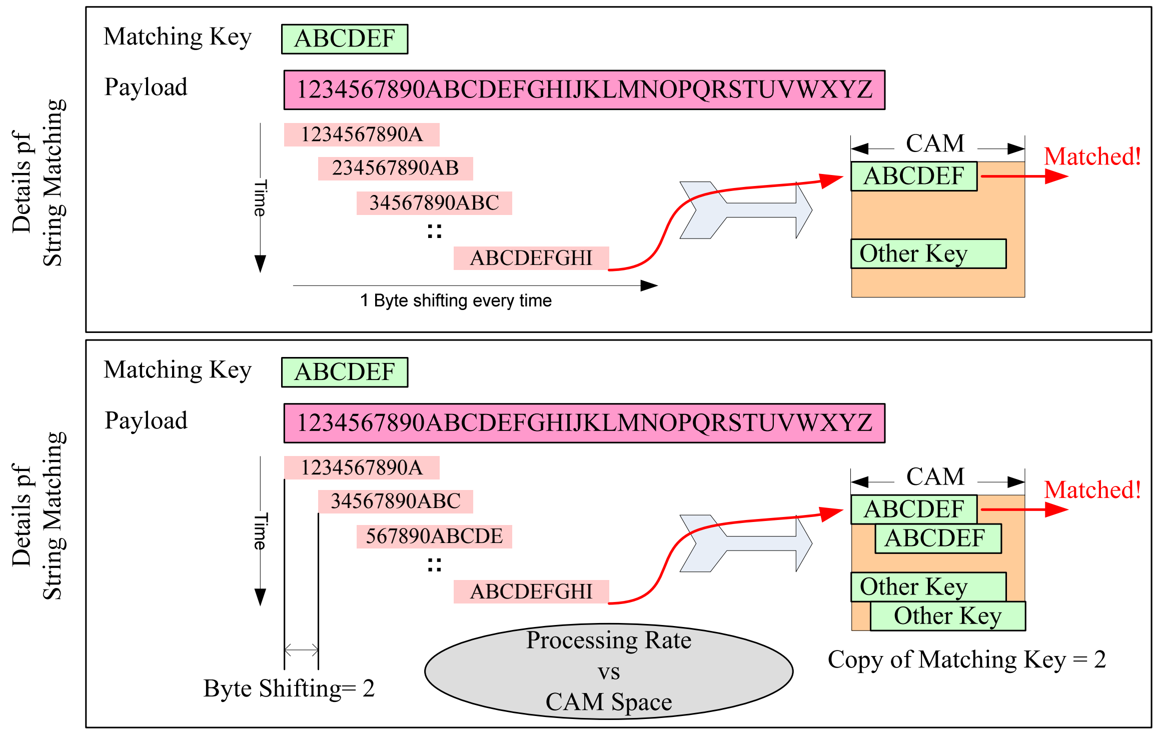


Figure 4.9 Single Short String Matching Method

Partial Solution for Requirements 5, 6, 7 and 8

There are cases when Snort payload criteria have other requirements in addition to just a single short string matching. These include:

- String Matching with Payload Offset range Criteria (16.7% - 32.0%)
- Matching of Multiple Independent Strings (7.0% - 10.0%)
- Matching of Long Strings (>26 Bytes) (3.3% -5.3%)
- Distance Range Matching Between Multiple Strings (12.0% - 27.5%)

In Repanshek's thesis [83], only partial solutions have been provided for these problems. Basically, these partial solutions use a CAM to handle the same simple single short string matching work, and an external post-processor will handle the remaining requirement in these tasks.

For string matching with payload offset range criteria, CAM will handle the string matching part, and the post processor will handle the offset range matching to see if the matched string occurs within a proper position range of the payload.

For multiple independent short string matching, CAM will handle each string matching, and the post processor will handle the one rule multiple string part. It will correlate the match result from CAM to check if all strings in one rule have been all matched.

For matching of very long strings that exceed the CAM width, the long strings can be broken into 2 or more strings that can fit within CAM. The chaining of these shorter strings can be processed and evaluated in the post-processor shown in Figure 4.10.

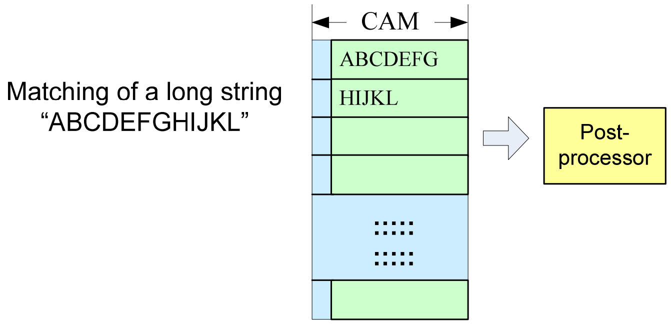


Figure 4.10 A CAM and Post-processor Chaining Method for Long String Matching

For multiple strings matching with a distance range between them, there are two possible solutions. One is a similar CAM plus post-processor solution where CAM will handle the string matching of these strings independently, with no consideration of distance range information. Distance range information can be processed in an external post-processor.

This CAM plus post-processor solution, along with the above solutions, has two performance issues: first, it uses a processor and thus slows down the fast processing rate of CAM. Second, since it removes the distance criteria from the CAM, it increases unnecessary partial match possibility and thus generates many unnecessary matches which compromise the CAM performance.

The other solution is a brute-force method that maps this criterion directly to the CAM shown in Figure 4.11. Using the "X (don't care)" bit in CAM, this payload criteria can be expanded to multiple CAM entries to list all possible distance ranges between these two strings. Any match in these expanded entries means a match to the payload criteria.

This method is memory space intensive. It has a width limitation based on the width of the CAM when the strings are long and the distance is big. It also has a linear depth expansion proportional to the range size of the allowed distance. This is not feasible for many rules where the distance range is large.

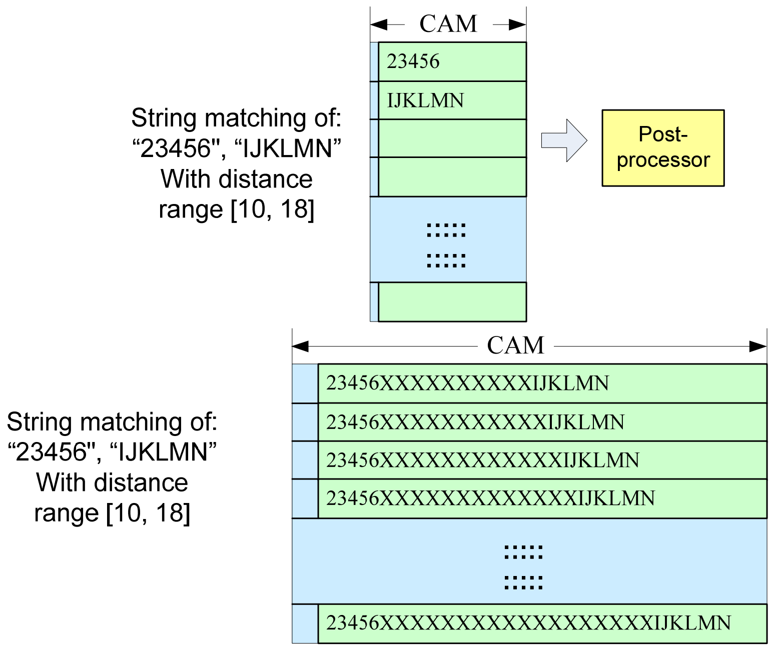


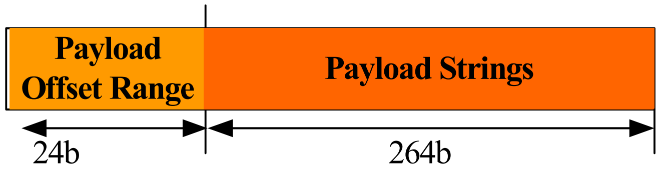
Figure 4.11 A CAM and Post-processor Multiple String Matchings with Distance Between Strings

4.3 NOVEL SNORT PAYLOAD STRING MATCHING WITH PAYLOAD OFFSET RANGE CRITERIA IN CAM

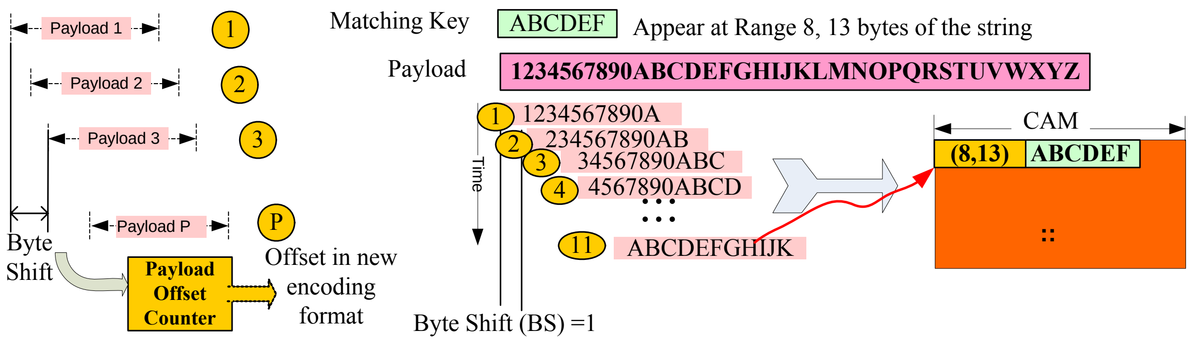
Problem: There are cases when not only the string has to be matched inside the payload, but the position of where the string appears in payload is also important. This string should be at a position within a specific range of payload. 16.7% - 32.0% of Snort Rules (version 2.0 through version 2.4) require this feature.

Example: A Rule has a string matching criteria for short string "ABCDEF" which should appear among the first 18-21 bytes of the payload. In Snort Rules, this rule will appear as "content: ABCDEF; offset:18; depth:3;".

Solution: For packet payload, range criteria in string matching are encoded using the encoding scheme for CAM range matching proposed in Chapter 3. These range criteria are mapped in CAM. The preprocessing work will count the packet payload byte shift to track the string position, and encode the string offset using the same encoding scheme. The format of mapping this string matching with a range criterion into CAM is shown in Figure 4.12.



(a) CAM Format for Key String with Offset Range in String Matching



(b) String Matching Process for Key Strings with Payload Offset Range Criteria

Figure 4.12 String Matching Method with Payload Offset Value

4.4 MATCHING OF MULTIPLE INDEPENDENT SHORT STRINGS

Problem: There are rules that have multiple independent short (<26B) matching criteria. A rule will be satisfied only when all these strings are found in the payload. 7.0% - 10.0% of Snort rules (version 2.0 through version 2.4) have this requirement.

Example: A Rule has a string matching criteria for short string "ABCDEF" and "12345" which may appear at different positions inside a packet payload. In Snort Rules, this rule will appear as "content: ABCDEF; content: 12345;".

Solution: These independent strings can be mapped to multiple entries in CAM. The match result of multiple entries in CAM can be combined together by control bits in the "action RAM" after CAM.

The format of control bits in the RAM is shown in Figure 4.13. Grey parts are not used here and thus can be ignored in this section. They will be used in the next section.

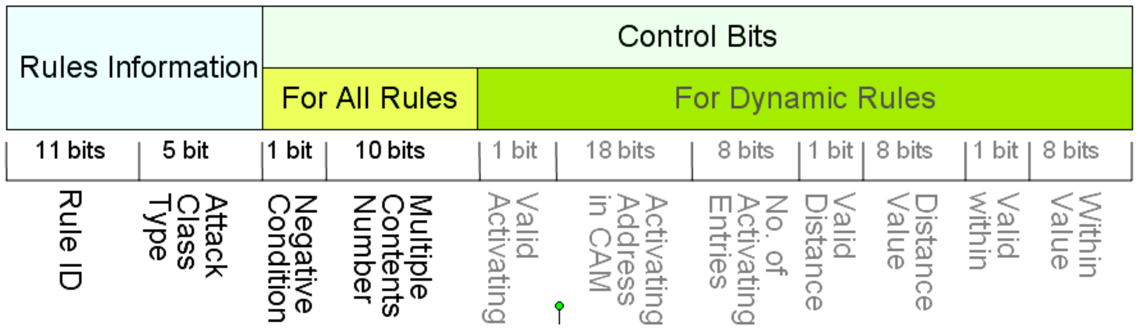
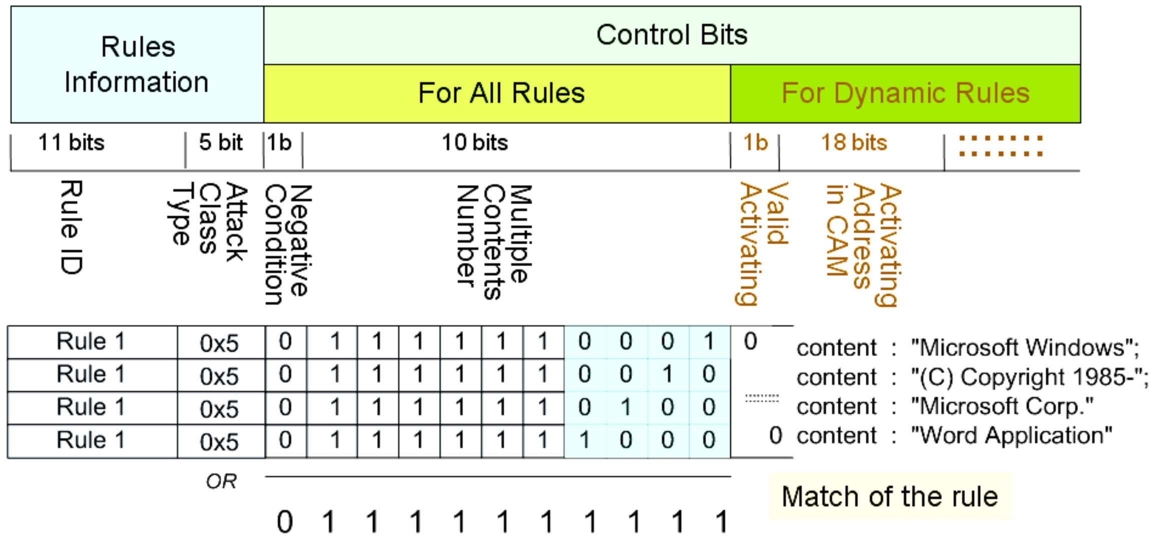


Figure 4.13 Control Bits in RAM

Each field of control bits are explained as below:

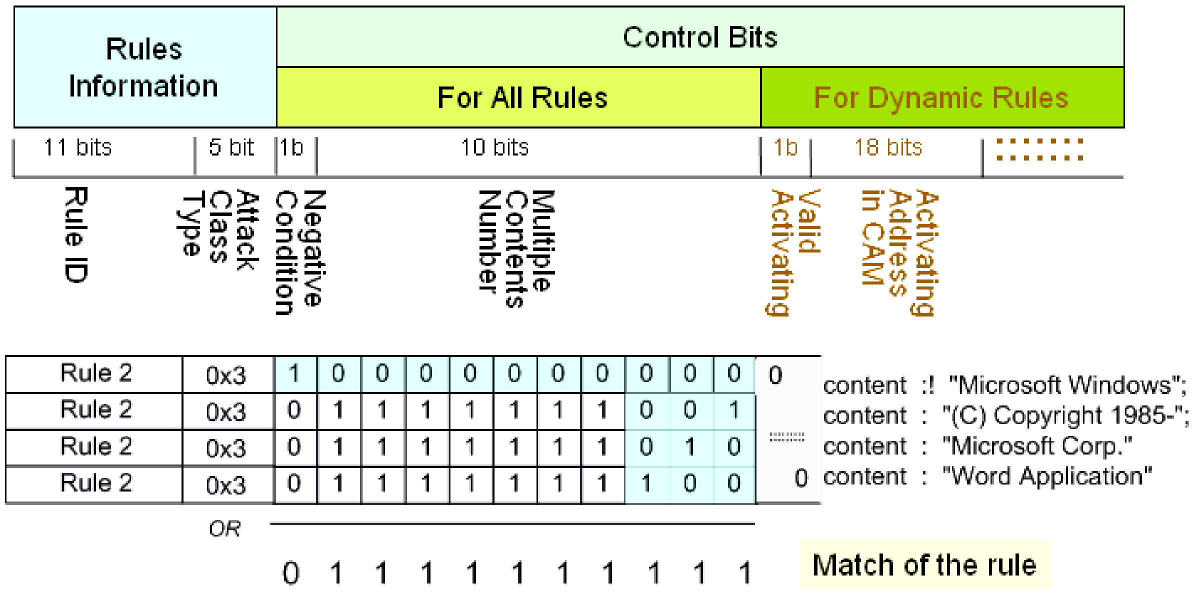
- Rule ID : Number of the rule from the .rules file
- Class Type : Attack Class Type field from the rule, totally 30 types
- Negative Condition Bit: whether this condition is negative. e.g. packet should not contain that particular content
- Multiple Content Number: which content is being checked for a multiple content rule

The control bits of "multiple contents number" are "OR"ed together after pattern matching of the whole packet is complete. If it goes to all '1', it means a match of the multiple string criteria rule. For example, as shown in Figure 4.14, if a rule requires that four strings are all existing in the payload, "Microsoft Windows", "(C) Copyright 1985-", "Microsoft Corp", and "Word Application", each of the strings will set only one '1' in the last four bits of "multiple contents number" field.



Rules Information
Control Bits
For All Rules
For Dynamic Rules
11 bits
5 bit
1b
10 bits
1b
18 bits
Rule ID
Attack Class Type
Negative Condition
Multiple Contents Number
Valid Activating
Activating Address in CAM
Rule 1 0x5 0 1 1 1 1 1 1 0 0 0 1 0 content : "Microsoft Windows";
Rule 1 0x5 0 1 1 1 1 1 1 0 0 1 0 content : "(C) Copyright 1985-";
Rule 1 0x5 0 1 1 1 1 1 1 0 1 0 0 content : "Microsoft Corp."
Rule 1 0x5 0 1 1 1 1 1 1 1 0 0 0 0 content : "Word Application"
OR
0 1 1 1 1 1 1 1 1 1 1
Match of the rule

(a) Example of Multiple Fields Control Bits



Rules Information
Control Bits
For All Rules
For Dynamic Rules
11 bits
5 bit
1b
10 bits
1b
18 bits
Rule ID
Attack Class Type
Negative Condition
Multiple Contents Number
Valid Activating
Activating Address in CAM
Rule 2 0x3 1 0 0 0 0 0 0 0 0 0 0 0 content :! "Microsoft Windows";
Rule 2 0x3 0 1 1 1 1 1 1 1 0 0 1 content : "(C) Copyright 1985-";
Rule 2 0x3 0 1 1 1 1 1 1 1 0 1 0 content : "Microsoft Corp."
Rule 2 0x3 0 1 1 1 1 1 1 1 1 0 0 0 content : "Word Application"
OR
0 1 1 1 1 1 1 1 1 1 1
Match of the rule

(b) Example of Negative Condition Control Bits

Figure 4.14 Example of Control Bits in Mapping Snort Rules

Another case occurs when the rule has a negative criterion that some string cannot exist in the payload. For example, if the first string "Microsoft Windows" should not appear anywhere above strings, this is a negative criteria. Negative criteria can be implemented by setting negative bit as ‘1’ and its “multiple contents field” all ‘0’. After the whole packet is checked, the control bits of all matched result are “OR”ed. Only if the result of “OR”ed negative bit is ‘0’ and all “multiple Contents Number” is ‘1’, will this whole rule be matched.

4.5 NOVEL MATCHING OF STRINGS LONGER THAN THE CAM WIDTH

Problem: Snort has different lengths of string matching. The length of the string in Snort version 2.0 is shown in Figure 4.15.

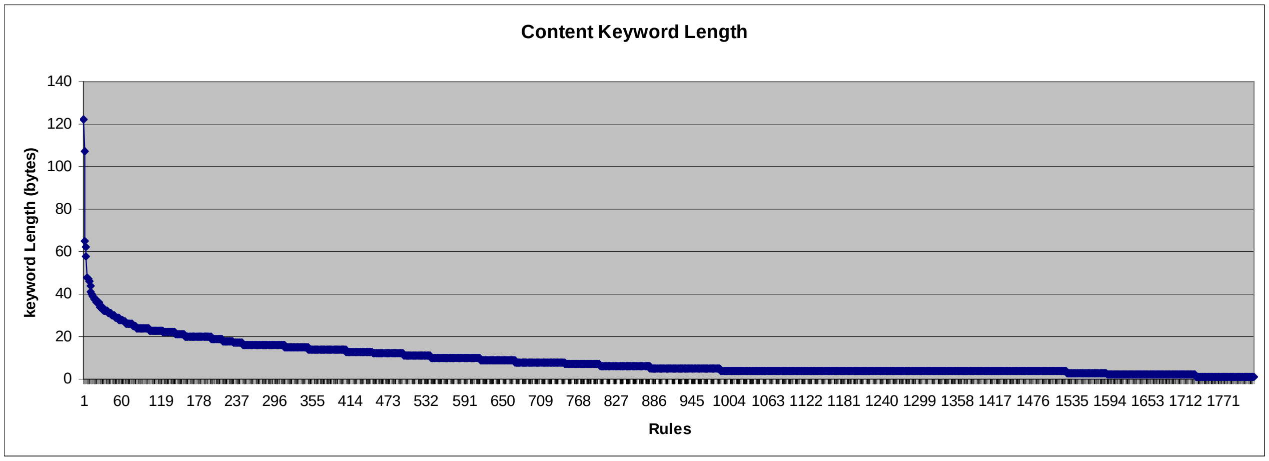


Figure 4.15 Keyword Length in Snort Payload String Matching

There are cases in packet payload criteria when the string matching is to be performed on very long strings, even longer than the CAM width allocated to payload string matching. 3.3% - 5.3% of Snort rules (version 2.0 through version 2.4) belong to this category. Long string matching cannot be mapped directly into a CAM.

Solution: We use the the “dynamic solution” for long string matching shown in Figure 4.16. It costs more than one searches just like the hardware/software chaining method.

However, the chaining information will be embedded in CAM instead of a post-processor. The search number in this method depends on how long the string is.

Using the long string “ABCDEFGHIJKL” as an example, the dynamic solution performs operations in the following steps:

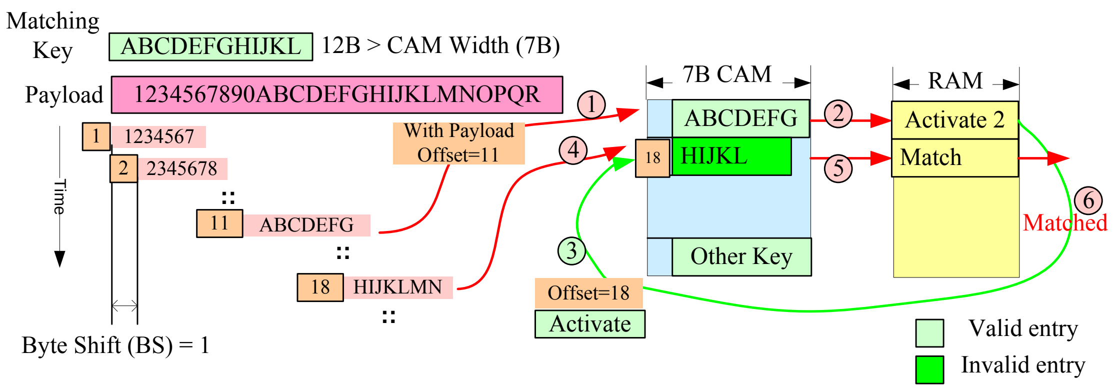


Figure 4.16 Dynamic Solution for Long String Matching

1. Initialization:

- Configure the CAM, with the first part of string (7B) in one CAM entry, and the second part string (5B) in the following CAM entry but invalidate the entry for now. Leave CAM bits in each entry for payload offset value criteria.
- Configure the RAM, with the action "Activate next CAM entry" in the address where the first string is in the CAM, and the action "match" in the following RAM entry.

2. Perform the following searches:

- First search:
   1. Segment of the payload searches the CAM. Assume it finds a match with the first string, with payload offset equals to 11.
   2. This match leads to the RAM address where an "activate the next CAM entry" command exists.
   3. Activate the next CAM entry with string 2 inside it, and at the same time, write the offset 11+7=18 into the entry.
- Second search:
   1. Segment of the payload searches the CAM. Assume it finds a match with the second string, with payload offset equals to 18. It should be noted, that both the string and the offset value need to be a match with that CAM entry.
   2. This match leads to the RAM address where a "match" indication exists to show the whole criteria with long string are matched.

The control information is embedded in the RAM for each rule. The gray part in Figure 4.13 is used here.

4.6 NOVEL DISTANCE RANGE MATCHING BETWEEN MULTIPLE STRINGS

Problem: There is one case in packet payload criteria where there are multiple strings to be matched and the distance between these strings should lie within a range. 12.0% - 27.5% of Snort rules (version 2.0 through version 2.4) have this requirement. Distance ranging matching cannot be mapped directly into a CAM.

Examples: The packet payload criterion can search two strings "12345" and "IJKLMN" in the payload and they should appear within the distance 10 to 18 bytes. The Snort rule checking this criterion can be written as: "Content: "23456"; Content: "IJKLMN"; distance: 10; within: 8;"

Solution: In the distance matching, 2/3 of the distances are bigger than 8 bytes. Only 1/3 of the distances are smaller than 8 bytes. We propose to use the above brute-force method for criteria with two strings no further than 8 bytes apart. This limits the depth expansion to less than 8 CAM entries.

For multiple strings with larger distances, the "dynamic solution" shown in Figure 4.17 is proposed which costs more than one searches but limits CAM space. The search number in this method is the number of multiple strings in the criteria.

Using the example with two strings, the dynamic solution operates in the following steps:

1. Initialization:

- Configure the CAM, with the first string in one CAM entry, and the second string in the following CAM entry but invalidate the entry for now. Leave CAM bits in each entry for payload offset value criteria.
- Configure the RAM, with the action "Activate next CAM entry" in the address where the first string is in the CAM, and the action "match" in the following RAM entry.

2. Search:

- First search:
    1. Segment of the payload searches the CAM. Assume it finds a match with the first string, with payload offset equals to 2.
    2. This match leads to the RAM address where an "activate the next CAM entry" command exists.

3. Activate the next CAM entry with string 2 inside it, and at the same time, write the offset range [2+10, 2+18] =[12, 20] into the entry.

- Second search:
  1. Segment of the payload searches the CAM. Assume it finds a match with the second string, with payload offset equals to 19. It should be noted, that both the string and the offset value need to be a match with the CAM entry.
  2. This match leads to the RAM address where a "match" indication exists to show the whole criteria with two strings are matched.

The control information is embedded in the RAM for each rule. The gray part in Figure 4.13 is used here.

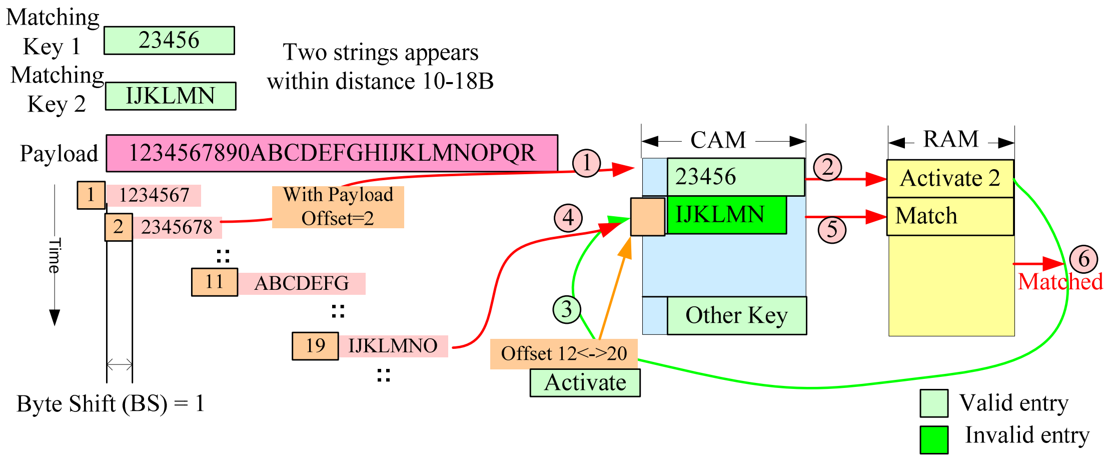


Figure 4.17 Dynamic Method of Multiple String Matching with Distance Between Them

4.7 DEMONSTRATION OF MAPPING FEASIBILITY OF SNORT

We demonstrate the feasibility of mapping Snort IDS in CAM hardware by using a prototype system shown in Figure 4.18. The architecture features meet requirements 1 through 8.

4.7.1 Mapping Snort IDS Rules in CAM

To completely match a Snort rule, both its header matching and payload string matching are required. The packet criteria will be mapped to limit width ternary CAM bit by bit. In each 576 bit wide CAM entry, the first 288 bits are allocated for rule header checking and the following 288 bits are allocated for payload string matching. Criteria for header fields checking will be directly compared with the incoming packet. Criteria for packet strings matching will be comparing with each shifting position of the packet payload, in order to implement the string matching function. The shifting bytes in string matching can be variable of one byte or multiple bytes, which trades search performance with CAM space as discussed in Figure 4.9. The action in the Snort rules and other control fields will be mapped into a RAM.

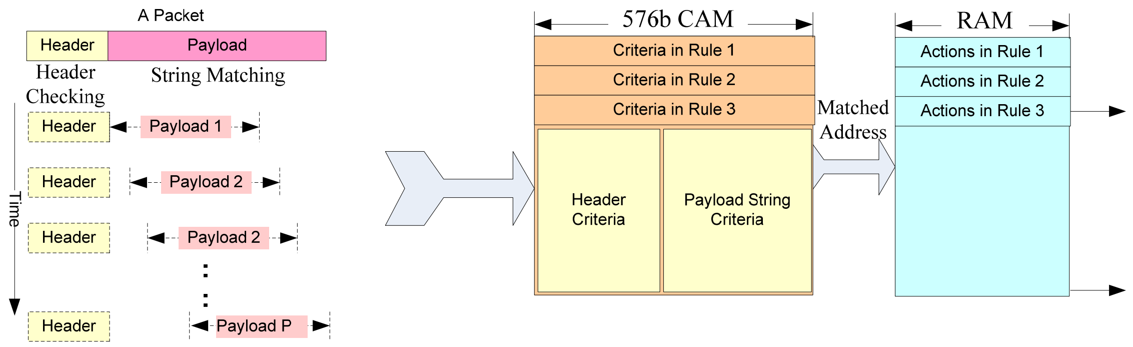


Figure 4.18 Putting Header Checking and Payload String Matching in One CAM Entry

4.7.2 Prototype System for CAM-based Snort

A prototype system is built in order to demonstrate the mapping Snort rules to a CAM. A CAM behavior model is established by SystemC language in order to take advantage of SystemC's cycle accurate simulation. The Snort rules are parsed through a Perl script to be mapped to this CAM using CAM initialization files.

Figure 4.19 shows the prototype system. A packet analyzer software program is made using C. This analyzer extracts proper fields from each packet and assembles fields in a corresponding format with the CAM rule format. Sniffer data containing network packets are analyzed by this program and fed into the CAM filled with Snort rules.

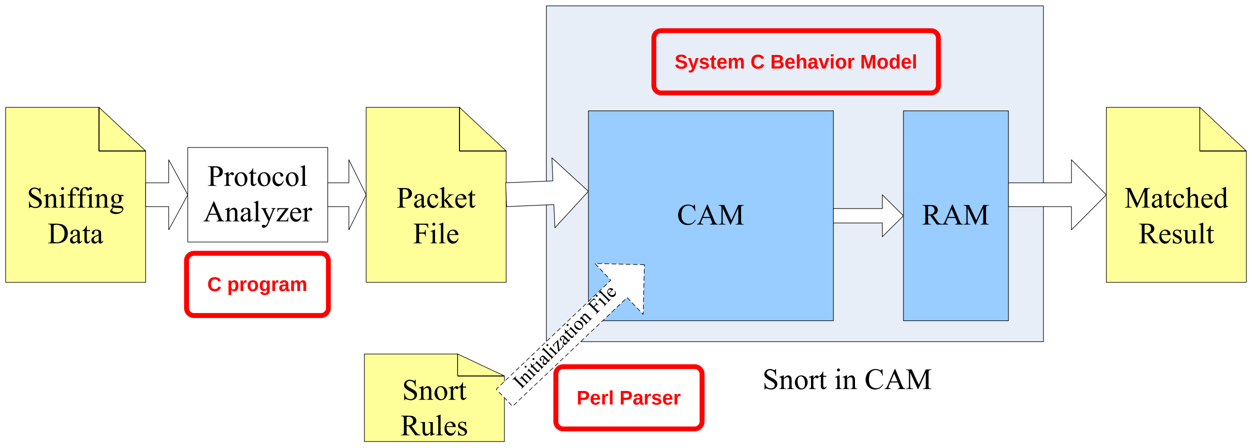


Figure 4.19 Prototype System for Single-Packet Attack Detection Using Snort Rules

4.7.3 Snort Rules Mapping Result in CAM

Using the above header matching and payload matching methods, we map Snort rules into a CAM. Perl script is used to automatically map the text file of Snort rules into CAM initialization files which contain the content of all CAM entries. Since there are some rules that are hard to map into CAM, e.g. rules with "Perl regular expression" or *threshold*, *byte_jump* and *byte_test* keywords, these rules are ignored and skipped for now. Figure 4.20 shows the number of mapped rules in version 2.0 through version 2.4 and the number of CAM entries it requires with simple 1 byte shift payload string matching. The detailed mapping results for Snort version 2.0 through version 2.4 are shown in Table 4.4. The mapping results are as follows:

- Use 1 byte shift of payload string matching, 1739 to 1809 Snort rules map into CAM as 2461 to 2569 CAM entries each 576 bits wide. This is 15.0% to 15.8% of the CAM space of a commercial IDT 75 series 9M CAM [85]. This mapping can achieve one complete search in 40 ns, equal to a search rate of 200Mbps for the packet.
- Use 9 byte shift of payload to do string matching, 1739 to 1809 Snort rules map into CAM as 20980 to 22177 CAM entries each 576 bits wide. This is 128% to 135% of the CAM space of a commercial IDT 75 series 9M CAM [85]. Since CAM can be cascaded to expand depth, these rules can be accommodated in two cascaded CAMs. This mapping can achieve one complete search in 40 ns, equal to a search rate of 1.8Gbps for the packet.

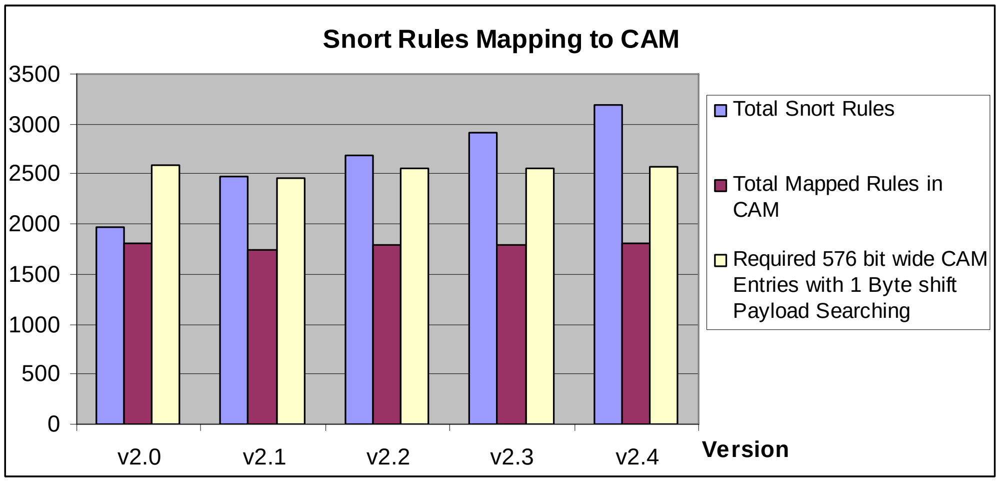


Figure 4.20 Number of Snort Rules (version 2.0 through version 2.4) and Required CAM Entries

Table 4.4 Mapping Results of Snort Rules (version 2.0 through version 2.4) To CAM

	Rules Mapped to CAM / Total Rules	**CAM Size: Number of 576b Wide Entries**	**Mapping Results with 1 Byte Shift Payload Matching**		**Mapping Results with 9 Byte Shift Payload Matching**	
			Number of 576b Wide Entries	**Percentage of a CAM size**	**Number of 576b Wide Entries**	**Percentage of a CAM size**
V 2.0	1802/1977	16384	2594	15.8%	22177	135%
V 2.1	1739/2474	16384	2461	15.0%	20980	128%
V 2.2	1793/2678	16384	2561	15.6%	21868	133%
V 2.3	1793/2911	16384	2561	15.6%	21868	133%
V 2.4	1809/3191	16384	2569	15.7%	21940	134%

4.8 CONCLUSION

This chapter introduced the detection of single-packet attacks using CAM, which forms the basic architecture of CAIPES. We use the Snort IDS rules version 2.0 through version 2.4 as examples. Four novel methods have been presented for mapping the Snort IDS rules into a CAM. They are as follows:

1. Use a novel encoding scheme for string matching with payload offset range criteria
2. Use RAM control bits for matching of multiple independent short strings
3. Use a dynamic method for matching of longer strings than CAM width
4. Use a dynamic method for distance range matching between multiple strings

These four methods solve 17.9%-28.4% of Snort rules, partly based on the new encoding scheme described in Chapter 3. A total of up to 91% of snort rules (version 2.0) have been mapped to the architecture. It achieves search rate as high as 1.8Gbps for network traffic. A prototype system has been developed for this architecture.

5.0 DETECTION OF MULTI-PACKET ATTACK

There are many complicated attacks launched in multiple packets. Twenty-eight attack types of the 58 network intrusions in the Lincoln Lab intrusion detection evaluation dataset [68] belong to the multi-packet attack category. Detection of multi-packet attack requires examining not only current packet information, but network context information left by previous packets as well. Therefore, incresaed detection ability using expert system features are required for these attacks.

We classify the 28 multi-packet attacks into 3 attack categories each requiring different detection abilities of the expert system. They are listed as follows:

- Flooding of packets attacks (29% of multi-packet attacks in Lincoln Lab data)
- Sequence of packets attacks (39% of multi-packet attacks in Lincoln Lab data)
- Flooding of sequences attacks (32% of multi-packet attacks in Lincoln Lab data)

A conventional expert system has a performance issue because of its computational nature in finding a match between two databases: the "fact base", which stores facts to be processed, and the "rule base" storing expertise knowledge. Conventional expert systems cannot handle fast network traffic at wire speed. Based on the features of network intrusions, we invented a novel method called "*Contextual Rules*" to convert the expert system structure into a one-database match problem, in the IDS/IPS problem domain. This Contextual Rules method dynamically combines current network states with intrusion rules and creates new Contextual Rules out of it. Searching the database of Contextual Rules equals to searching both the fact base and the rules base to find a match.

Content Addressable Memory (CAM) [77] is a parallel search engine chip that searches its memory contents in parallel to find one or more entries that are equal to the input data. It has been proposed to accelerate many network functions that are search-intensive by nature, such as packet classification and intrusion detection. Unlike software-based solutions that perform serial searches and cost linear search time, a CAM performs a search operation in parallel and takes a constant number of clock cycles, typically tens of nanoseconds in current technology. This is 100 times to 1000 times faster than software running on processors. In addition, the search time is independent of the number of rules used in a given search. This performance motivates its application in CAIPES.

Using a parallel search engine Content Addressable Memory or CAM, the Contextual Rule database can be searched in parallel within a single clock cycle. Therefore we can apply the parallel search power of CAM to the expert system matching problem. Based on the CAM-based single-packet detection introduced in Chapter 4, we designed our CAM-Assisted Intrusion Prevention System (CAIPES).

In Section 5.1, we introduce the basics of the Lincoln Lab IDS Evaluation dataset, and classify the multi-packet attacks into three categories. In Section 5.2, we discuss expert system architecture and its performance problem in IDS/IPS problem domain. In Section 5.3, we present a novel method called *Contextual Rule*s to convert the expert system architecture into one-database search architecture. We then show how CAM's parallel search capability can be applied to this architecture. In Section 5.4, we discuss design issues and questions involved in the Contextual-Rule Method including the mapping feasibility of Contextual rules to CAM, as well as its performance. In Sections 5.5 to 5.9, we present the design details of Category 1 through Category 4 detection respectively. Simulation results from each category's detection are presented and discussed. Section 5.9 discusses the scalability problem of CAIPES as well as a buffer overflow protection mechanism. We conclude this chapter in Section 5.10.

5.1 MOTIVATION

5.1.1 Lincoln Lab IDS Evaluation Data Introduction

IDS evaluation research led by Lincoln Lab in 1998 and 1999 provides offline evaluation data corpus to measure the correctness in IDS detection in aspects of detection ability [68]. We use it to help determine what detection abilities an IDS should have, as well as in the test of our CAIPES' detection ability on multi-packet attacks.

Lincoln Lab data corpus are presented in the forms of network sniffer data, file dumps, as well as audit data, in order to be used in both network-based and host-based IDS. Since CAIPES is a network-based IDS, we use the network sniffer traffic data as our object of study. Evaluation has two types of data [68]: 1) attack-free data to train the anomaly detection; 2) testing data containing both old and novel intrusions.

The details of network traffic in this IDS dataset are as follows [68]: a test bed network simulating live traffic between a small air Force base and the Internet has been implemented by Lincoln Lab as shown in Figure 5.1. Background traffic is simulated as if occurring from hundreds of programmers, secretaries, managers and other kinds of users running UNIX and Windows NT. Remote attackers launch attacks from five attacking machines (in the dark grey box), against the CISCO router and the four victims (in the light grey box). Sniffer data is collected from both outside traffic and inside traffic using the tcpdump utility.

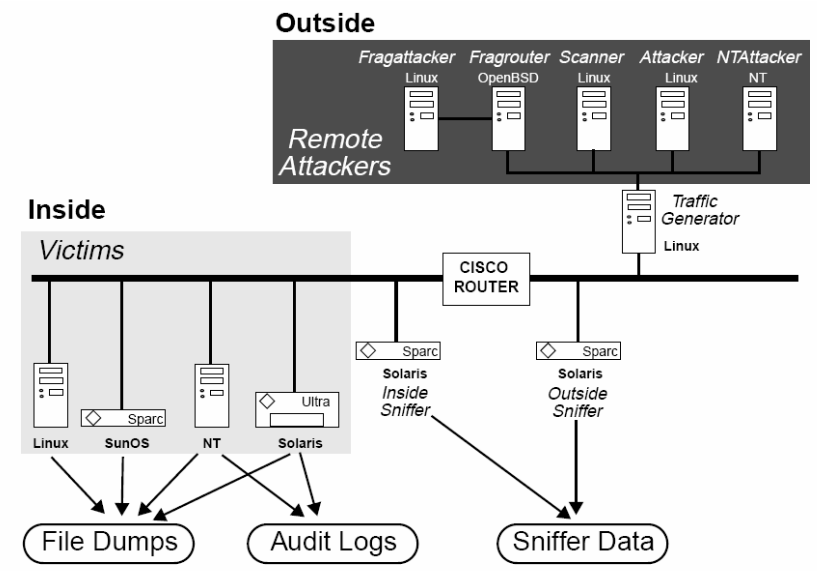


Figure 5.1 Simulating Network in 1999 Lincoln Lab DARPA IDS Evaluation (from [68])

5.1.2 Attack Classification According to Expert System's Detection Ability

There have been various classifications of attacks proposed from different perspectives. This thesis introduces a new method of classifying the 58 attacks in the 1999 Lincoln Lab dataset. This classification is from an expert systems' detection perspective, with attacks in the same category sharing similarity in their detection methods. Four categories of attacks ranging from single-packet attack to complex multiple-packets attack are listed in Table 5.1. Categories 2 through 4 require multi-packet detection that can be fulfilled by the extra detection capability of expert systems beyond the single-packet detection methods discussed in Chapter 4.

Table 5.1 Four Categories in 1999 Lincoln Lab Data of Network Attacks

Categories		**Intrusion Features**	**Enumerated Attack**	**Number of Attacks**
Single-Packet Attack	Category 1	Single packet	UDPStorm CrashIIS Casesen Eject Ffbconfig Fdformat Perl Ps Sechole Xterm Yaga Imap Named NcFTP Netbus Phf Secret Httptunnel Ls_domain Land Sendmail SelfPing Loadmodule Syslogd	24
Multi-Packet Attack	Category 2	Occurrence frequency of a legitimate packet	Ping of Death Apache2 Smurf UDPStorm IPsweep Back Mailbomb Dictionary	8
	Category 3	The specific sequence of multiple packets	Dosnuke TCPReset Ftpwrite Queso Saint Satan ARPPoison TearDrop Netcat Xlock Resetscan	11
	Category 4	Occurrence frequency of a packets sequence	SYN Flood (Neptune) SshProcessTable Insidesniffer Processtable Nmap Guest NTinfoscan Anypw Mscan	9

*There are six types of attacks in Lincoln Lab data that are not detectable in network traffic.

Category 1 Single-Packet Attacks: Attacks in this category all consist of only one specific packet. Finding this specific packet in the network means the attack has taken place or will take place. To detect Category 1 attack, the expert system NIDS should examine each incoming packet, and check the header information and payload string to see if the current packet matches any signature of attacks in Category 1. This has been solved using the same methods as in Chapter 4.

Category 2 Flooding of Packet Attacks (29% of multi-packet attacks): Attacks in this category consist of flooding of a legitimate packet by its own. Finding this specific packet occurring in the network at a high frequency means the attack has taken place or will take place. To detect Category 2 attacks, the expert system NIDS should examine each incoming packet to see if the current packet belongs to any of the specific packets in Category 2. If it does, the expert system will increment a counter which keeps track of the number of occurrences during a sliding window of time. Any number bigger than a threshold means a Category 2 attack has happened.

Category 3 Sequence of Packets Attacks (39% of multi-packet attacks): Attacks in this category consist of a specific sequence of multiple packets. This sequence can be an attack scenario or a protocol transition sequence. Finding that this specific packet sequence occurred in the network within a time period means the attack has taken place or will take place. To detect Category 3 attacks, state machines of various sequences will be built inside the expert system. The expert system NIDS should examine each incoming packet to see if the current packet belongs to any packets in the specific sequences in Category 3. If it does, the expert system will determine from the current state in the state machine to see if it should advance the state machine of this packets sequence. Completing a state machine means a Category 3 attack has happened.

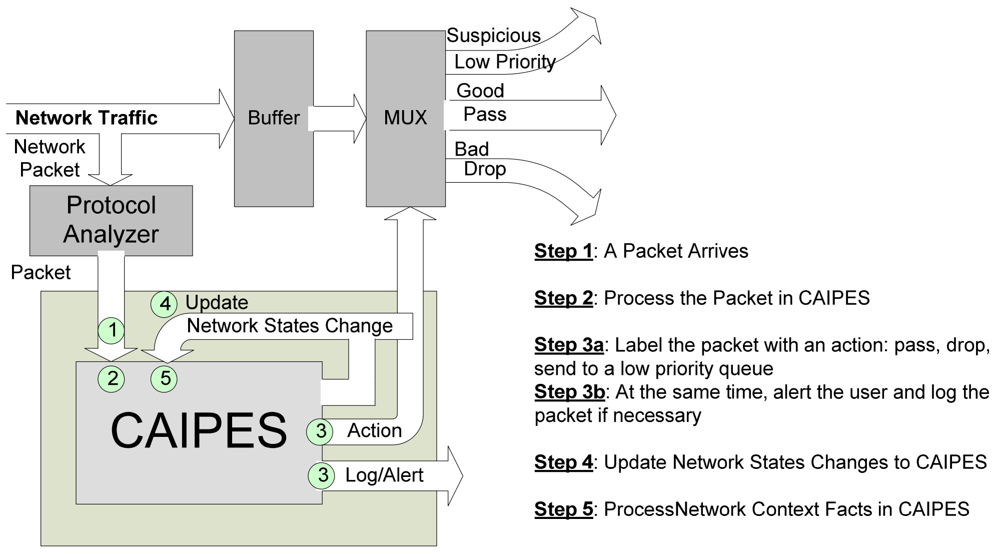
Category 4 Flooding of Sequences Attacks (32% of multi-packet attacks): Attacks in this category consist of flooding of a specific packets sequence similar to category 3. Finding this specific packet sequence occurring in the network at a high frequency means the attack has taken place or will take place. To detect Category 4 attacks, state machines of various sequences should be built inside the expert system. The expert system NIDS should examine each incoming packet to see if the current packet belongs to any of the specific packets in Category 4. If it does, the expert system will determine from the current state in the state machine to see if it should advance the state machine of packets sequence. If a state machine is completed, the expert system will increment a counter which keeps track of the number of occurrences during a sliding window of time. Any number bigger than a threshold means a Category 4 attack has happened.

5.2 INTRUSION PREVENTION EXPERT SYSTEM

An expert system is defined as “a computer program that represents and reasons with knowledge of some specialist with a view to solving problems or giving advice” [41]. An expert system has two parts: a “fact base” storing facts to be processed and a “rule base” storing expertise knowledge. The expert system reasons and makes decision according to the current facts and rules.

An IDS/IPDS expert system works as follows: network traffic is sent to the expert system, packet by packet. The fact base in the expert system stores these packets, as well as various network states, for example, the intermediate attack state and traffic statistics. The rule base in an expert system stores expert knowledge from human experts, represented in “IF, THEN” format. The part after “IF” is the “condition” part where current facts are compared to see if they satisfy the condition or not. The part after “THEN” is the “action” part which the expert system should take when the conditions are satisfied. In this way, the expert system analyzes the traffic and decides whether it’s good, bad, or suspicious. The expert system then takes actions to pass, drop, or send packets to a low priority queue respectively.

This research proposes a unified IPS expert system architecture that uses a highly parallel search memory to perform intrusion detection and prevention at Gigabit wire speed. This unified CAM-Assisted Intrusion Prevention Expert System (CAIPES) architecture supports various detection methods, as well as prevention actions. The actions includes stopping malicious traffic, allowing good traffic, sending suspicious traffic to lower priority queues, and logging, as shown in Figure 5.2. For example, in Step 1, a new packet arrives. It is parsed by protocol analyzer and then fed into CAIPES. In Step 2, CAIPES process the packet with its expert knowledge and determine whether the packet is good or bad. In Step 3, CAIPES system sends out the decision results, including labeling the packet with an action of being “passed”, “dropped”, or “sent to a low priority queue”, as well as messages to alert the user and log the packet if necessary. In Step 4, any network state changes due to this packet are fed into CAIPES. These changes include TCP connection status change, traffic statistics change. Finally, in Step 5, the changed network states are evaluated in the CAIPES system again. Steps 1 to 5 are then repeated again.



Suspicious
Low Priority
Good
Pass
Bad
Drop
Network Traffic
Network Packet
Buffer
MUX
Protocol Analyzer
Packet
4 Update
Network States Change
1
2
5
CAIPES
3 Action
3 Log/Alert
Step 1: A Packet Arrives
Step 2: Process the Packet in CAIPES
Step 3a: Label the packet with an action: pass, drop, send to a low priority queue
Step 3b: At the same time, alert the user and log the packet if necessary
Step 4: Update Network States Changes to CAIPES
Step 5: ProcessNetwork Context Facts in CAIPES

Figure 5.2 Function Diagram of CAIPES

The novelty of this CAIPES architecture is the application of CAM's parallel search capability to expert systems' match processes, thereby accelerating the expert systems for network IPS application. The major contribution of this architecture is the mapping of an expert system's "match problem" into a CAM, and thereby using CAM parallelism to accelerate the expert system execution for IDS/IPS.

5.2.1 Expert System for IDS/IPS Domain

For a specific problem domain such as IDS/IPS, there are features in its facts and rules for an expert system that can be used to simplify the system execution and optimize the system performance. These features are not common to general expert systems. We observed four features in IDS/IPS monitoring process:

1. There are two kinds of facts that an expert system would look for to decide if traffic is malicious:
   - We define *Packet Facts* as network packets coming into the expert system directly.
   - We define *Context Facts* as current network states resulting from previous packets. These include protocol transition state, TCP connection state, intermediate attack state and all traffic statistics. Context Facts are results from previous packets but are not from outside the expert system directly.

2. Network attacks are caused by network packets and thus the Packet facts are the "driving" facts in detecting attacks. Context facts provide the network contexts in which an attack happens. Detecting an attack includes detecting a specific incoming packet under a particular network context.
3. All packet facts are discarded after they are processed. Context facts are kept unless an activated rule demands it be deleted, added or modified.
4. Variable binding among facts demanded by rules can be performed on the packet fact and context facts, for example, by checking the current packet IP address and port number in both the packet fact and the context facts. Variable binding between two or more context facts can be dissolved into two or more bindings each between the packet fact and context facts respectively.

Figure 5.3 shows the match process of an expert system for IDS/IPS. The Fact Base contains two kinds of facts: multiple context facts that records network states, and one single-packet fact that occurs when the current packet reaches the IDS. The Rule Base contains rules that each have two parts: a condition part with an "IF" clause to check whether the current fact base meet its criteria or not, and an action part with a "THEN" clause to state what to do when the condition is met. The matching process in expert system takes between two databases, the fact base and the rule base. It evaluates each condition in each rule with current facts to see if they are satisfied or not. The matching process is the major performance bottleneck in expert systems.

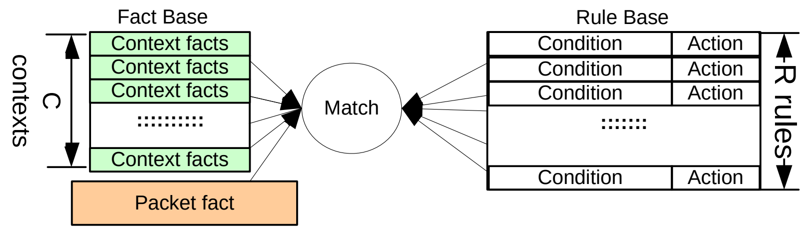


Figure 5.3 Details of Expert System Fact Base for the Match Process

5.2.2 The Performance Issue of Expert System Match Process

In a conventional expert system, matching occurs between the fact base and the rule base, as shown in Figure 5.3. Assume there are one packet fact, *C* context facts, and *R* rules in these two databases. The brute force matching algorithm has *O(C * R)* complexity. The match process can be described in the following pseudo code:

```
For (i=1; i<=R; i++)    // Evaluate rules one by one
{
    Check if Packet_Fact satisfy Rule[i];
    For (j=1; j<=C; j++)  // Check each context fact;
    {
        Check if Context_Fact[j] satisfy Rule[i];
    }
}
```

Assuming we have 100 context facts in a network, and 100 rules, the matching process has a total of 100*100 evaluation times. In a Giga Hertz processor, this takes about 100,000 clock cycles thus 100,000 ns for just one matching execution. This execution time is far beyond the minimum packet interval of 500ns in Gigabit networks and couldn't keep up with the traffic at wire speed. The performance deficiency of conventional expert system has motivated this acceleration research.

5.3 DYNAMICALLY JOINED CONTEXTUAL-RULES METHOD FOR EXPERT SYSTEM

5.3.1 Expert System Rules in IDS/IPS Domain

For the IPS domain, expert rules are attack signatures describing the characteristic of intrusions. An expert rule has two parts: conditions and actions. In the conditions part, there are three types of criteria in a signature:

1. *Packet criteria* to be applied on packet fact, for example, check if the packet protocol is TCP, and the TCP flag SYN and ACK bits are set.
2. *Context criteria* to be applied on context facts, for example, check if the network has TCP connection state is state 1.

3. Variable binding *Inter-Factual criteria* to be applied between packet fact and context facts, shown as *In Between* criteria in Figure 5.4. For example, check if packet fact has the same IP address and port number pair as that of the TCP connection state.

The first part packet criteria are present for all rules. The second and third parts are optional and depend on different detection types. Figure 5.4 shows the features of facts and rules in expert system in the IDS/IPS domain. There are two kinds of facts in the fact base: packet fact in orange and context facts in green. There are three parts in a rule's condition: packet criteria in orange, context criteria in green and variable binding criteria in yellow.

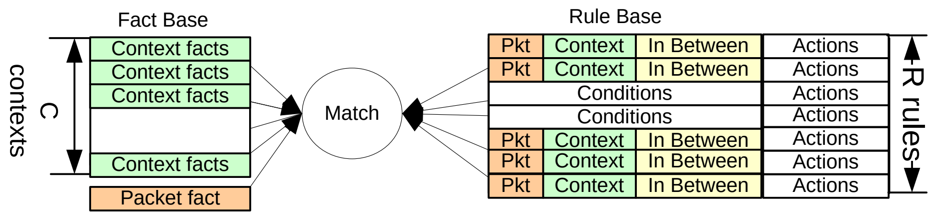


Figure 5.4 Details of Match Process in Expert System for IDS/IPS

5.3.2 Contextual Rules Method in Expert System in IDS/IPS Application

We invented a novel "Contextual Rules" (C-Rule) method that convert the two database search problem in expert systems to a one database search problem in the IPS domain. This "Contextual Rule" method dynamically combines current network states with intrusion rules and creates new Contextual Rules out of it. Contextual Rules are dynamics rules that only exist when all its context criteria are met under currently network context. As shown in Figure 5.5, Contextual Rules contain packet criteria and inter-factual criteria for any incoming packet to match. Inter-factual criteria usually checks whether the parameter in one fact equals to the parameter in another fact: for example, it checks whether the IP address fields in the Packet fact equal to a Context fact's IP address fields.

To implement the Contextual Rule method, the expert system breaks the matching process into two parts:

1. Pre-join matching between network context facts and rules' context criteria;
2. C-rule matching between packet fact and rules' packet criteria plus inter-factual criteria.

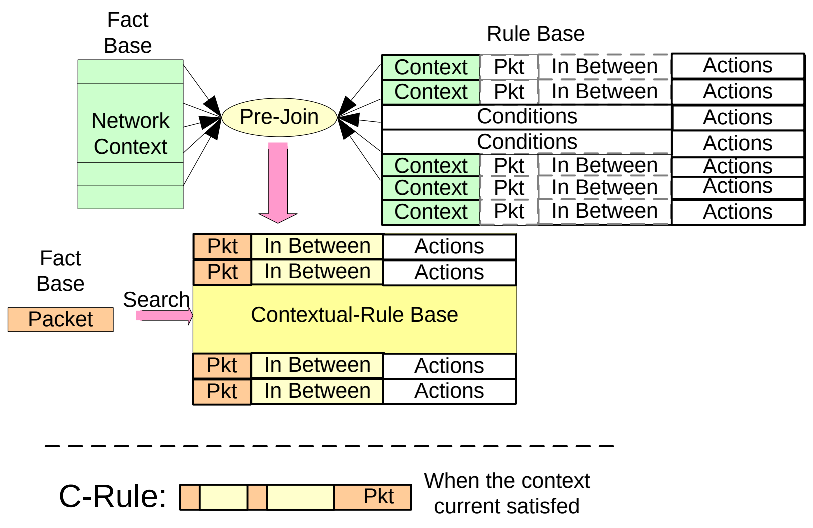


Figure 5.5 Contextual Rule Method

Therefore, the C-Rule method joins the expert system rule base with the fact base to get dynamic contextual rules according to the current network context. Therefore, the presence of a Contextual Rule already implies certain network context information. Each C-Rule consists of the packet criteria and the binding criteria between the packet and the current context. These C-Rules can be used to pattern match against incoming packet, without considering any context facts again. C-Rule base storing C-Rules will be searched directly by the network packet fact.

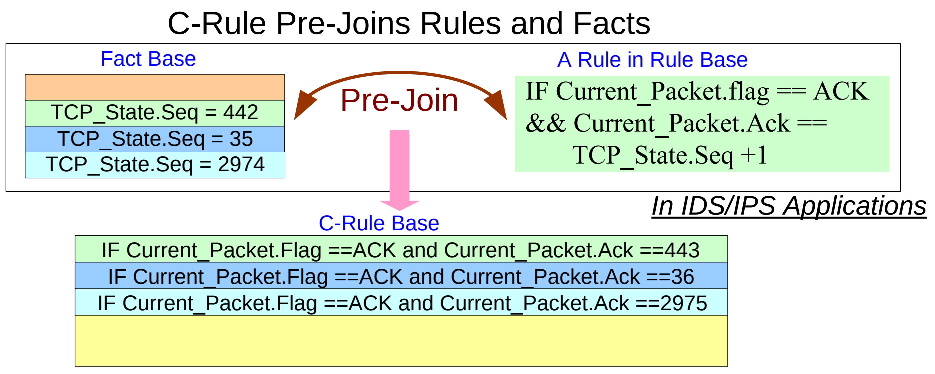

C-Rule Pre-Joins Rules and Facts
Fact Base
TCP_State.Seq = 442
TCP_State.Seq = 35
TCP_State.Seq = 2974
Pre-Join
A Rule in Rule Base
IF Current_Packet.flag == ACK && Current_Packet.Ack == TCP_State.Seq +1
In IDS/IPS Applications
C-Rule Base
IF Current_Packet.Flag ==ACK and Current_Packet.Ack ==443
IF Current_Packet.Flag ==ACK and Current_Packet.Ack ==36
IF Current_Packet.Flag ==ACK and Current_Packet.Ack ==2975

Figure 5.6 An Example of Pre-Join in the "C-Rule" Method

An example of the C-rule method is shown in Figure 5.6. The rule is part of a TCP state transition rule set. It checks if a future packet will have the "acknowledge" flag set to move the current TCP state forward. In the current fact base, there are three TCP states that can be progressed to the next state. The Pre-join matching matches the rule and facts, resulting in three new Contextual Rules based on the current facts with a very specific acknowledge number that the incoming TCP packet should carry in order to move the TCP state forward.

5.3.3 Optimization of Contextual Rules Method

An optimization to the Contextual Rules method is introduced here. We optimize the “Pre-join” matching process by putting the related rule and facts number in each Contextual Rule’s action part, as shown in Figure 5.7. This is feasible because when this C-rule is a match with an incoming packet, new context facts will be generated from this match and added into the fact base. These new context facts will only trigger a limited number of rules if a limited number of other facts exist. These rules and facts are the related facts and rules indicated in the C-rule’s action’s part. Therefore, Pre-join doesn't need to search the whole rule base and fact base to generate new C-rules. It can search only related rules and related facts stated in the match C-rule to perform the “Pre-join” match. It is similar to the Rete network where only changed facts but not all facts in fact base are compared with the rule base every time. Related facts and rules can possibly be only a small percentage of the total content in the fact base and rule base. This can significantly reduce the matching complexity of the Pre-join process.

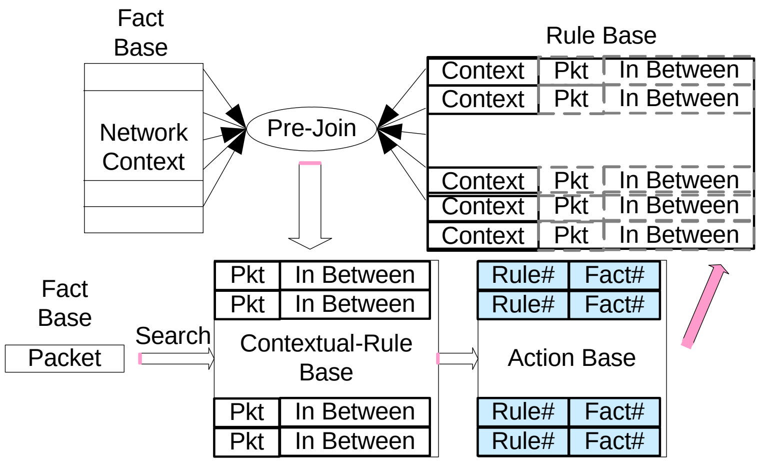


Figure 5.7 C-Rule Method with “Related Facts and Rules” Optimization

Therefore, the detection process in a C-Rule based system occurs as follows:

1. A network packet comes in, and searches the C-Rule base
2. If there is a match in the C-Rule base, actions corresponding to this rule are executed, they include modifying network contexts and sending an alert message if necessary.
3. The modified network context will go into the “Pre-join” block, trigger a “joining” process to activate any new C-Rules under current network contexts. The “related

facts" and "related rules" fields appended to the matched C-Rule will accelerate the "Pre-join" process.

4. New C-Rules are added into C-Rule base. The system waits for the next incoming packet.

Some of the modified contexts of step 3 do nothing other than generating a new C-Rule. These contexts are only used as the only criterion in one rule, and are not related to other rules. In this case, the consequence of that context fact is to activate that rule into a new C-Rule. There is no need to store this context in the fact base for future use. We can add the "activating rule number" field directly after the "related rule" and "related fact" field in that C-Rule as shown in Figure 5.8, in order to activate the proper new C-Rule without the Pre-join process. This "activating rule number" field is optional and can be dismissed when no rule can be directly activated from this changed context.

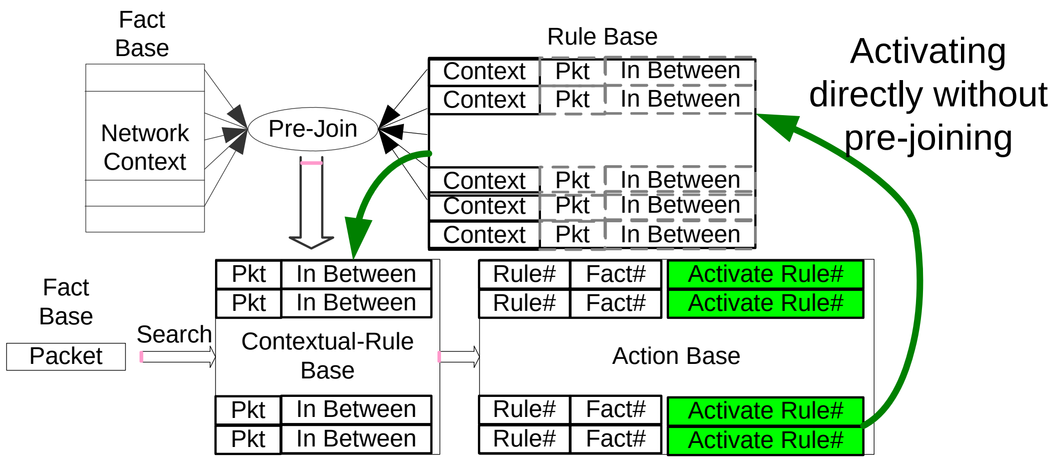


Figure 5.8 C-Rule Method with "Activated Rules" Optimization

In this case, the C-Rule based detection is simplified to the following steps:

1. A network packet comes in, and searches the C-Rule base.
2. If there is a match in the C-Rule base, actions corresponding to this rule are executed, they include modifying network contexts and sending an alert message if necessary.
3. The modified network context will activate new C-Rules which are added into the C-Rule base. The system waits for the next incoming packet.

5.3.4 CAM Assisted Contextual Rules Method

The novelty of the CAIPES architecture is the application of CAM's parallel search ability to expert systems' match processes, thereby accelerating the expert systems for network IPS application. The major contribution involves mapping an expert system's match problem into a CAM, and thereby using CAM parallelism for the expert system execution for the IDS/IPS problem domain.

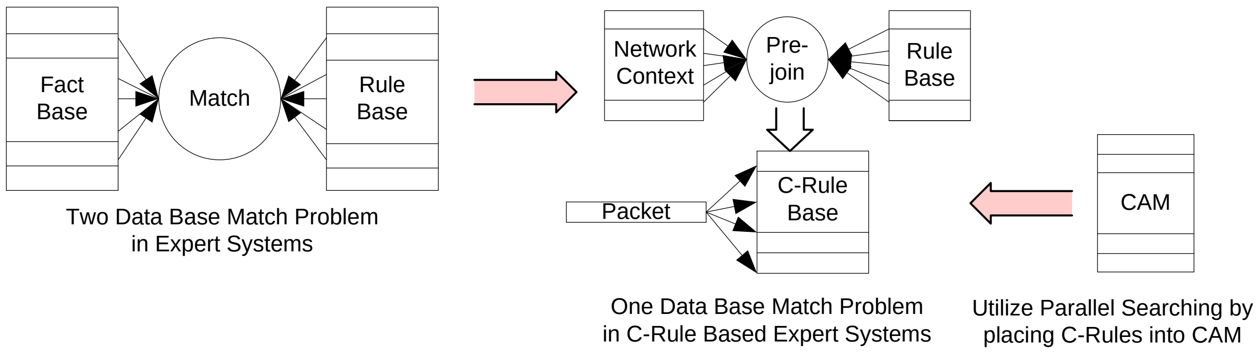

Fact Base
Match
Rule Base
Two Data Base Match Problem in Expert Systems
Network Context
Pre-join
Rule Base
C-Rule Base
Packet
CAM
One Data Base Match Problem in C-Rule Based Expert Systems
Utilize Parallel Searching by placing C-Rules into CAM

Figure 5.9 Application of CAM in C-Rule Based Expert System

The parallel search operation in a CAM is a parallel match process inside one database. It allows one pattern to search all entries of a database simultaneously. Therefore, the C-rule method converts the expert system two-database search problem into a one-database search problem, and this one database can be fit into a CAM to exploit CAM's density and search parallelism, as shown in Figure 5.9. The implementation diagram of a C-Rule method in CAIPES architecture is shown in Figure 5.10. The C-Rule base is expected to have more entries than the original rule base, as a result of joining the rule base and the fact base. Expanding depth in the CAM can be satisfied with current CAM technology. One CAM has tens of thousands entries (e.g. one IDT CAM has 16K * 576b entries) and multiple CAMs can be cascaded to have more entries [85].

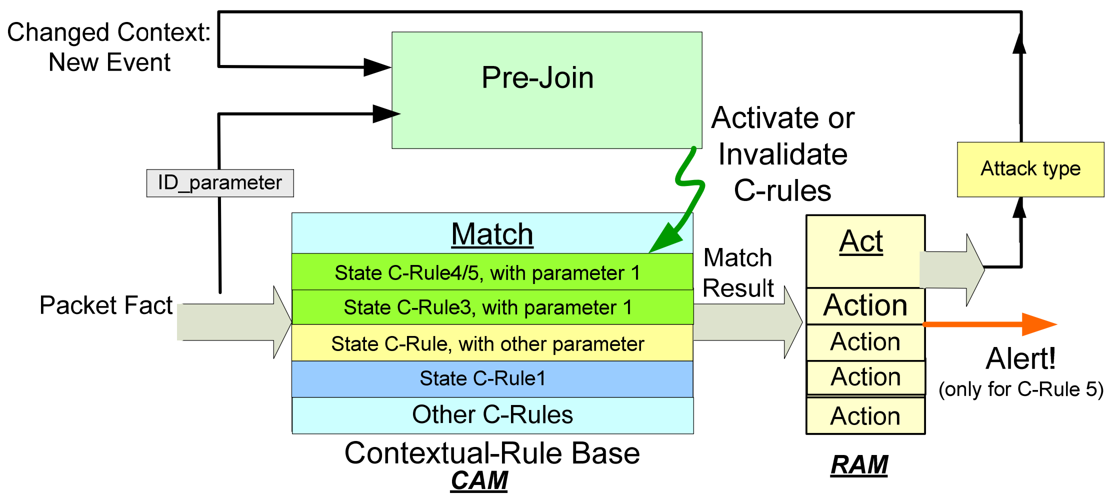


Figure 5.10 C-Rule Based Expert System with CAM

5.3.5 Match Complexity of C-Rule Method

In the Contextual Rule based method, matching happens in two places:

1. Pre-join matching between network context facts and rules' context criteria;
2. C-rule matching between packet fact and rules' packet criteria plus inter-factual criteria.

In Pre-join matching, an optimization to the Contextual Rules method is proposed to only search related rules and related facts stated in the match C-rule. This can significantly reduce the matching complexity of the Pre-join process. Before the optimization, the Pre-join matching complexity is $O(C*R)$ where C is the total number of contexts in the pre-join block and R is the total number of rules. The complexity of optimized Pre-join matching is $O(C_{(related)} * R_{(related)})$ where $C_{(related)}$ is the related contexts number to this context, and $R_{(related)}$ is the related Rule number to this context. Another further optimization has valid "Activating Rules" field appended to the matched C-Rule to directly activate/invalidate new rules. This avoids the pre-join process and the matching complexity is a O(1).

In the C-Rule matching, CAM is used for parallel matching with incoming packets. The matching process is still irrelevant to the number of current facts or rules. The match complexity is $O(1)$.

5.4 ISSUES IN THE C-RULE METHOD

In order for the C-Rule method to be feasible and to have wire speed detection performance in fast networks, there are several issues need to be delved into and solved. These are the assumptions and requirements for a C-Rule system to provide sufficient functions and satisfactory performance. The issues of concern are listed as follows:

Issue 1: Mapping C-Rule's Condition Evaluation into a CAM

Issue 1 focuses on the mapping feasibility of C-Rule's condition part into a CAM. What is the matching ability of CAM? In the C-Rule method, what kind of condition will a C-Rule have? Can CAM's matching ability handle the entire C-Rule condition matching requirements? We address this in Section 5.4.1 where details of how C-Rules can be mapped into a CAM are explained.

Issue 2: Mapping C-Rule's Action

Issue 2 focuses on the implementation feasibility of the C-Rule's action part in the C-Rule method. What kind of actions does a C-Rule have? How can these actions be implemented in the CAIPES system? We address this issue in Section 5.4.2 which explains how the details of C-Rule actions can be mapped into a RAM.

Issue 3: Performance of CAIPES Latency and Performance

Issue 3 focuses on the performance of the proposed CAIPES. How fast can CAIPES process network packets? Is it faster than conventional expert system? What affects its performance? We address this issue in Section 5.4.3 where details of high level performance analysis and comparison with conventional expert system are shown.

Issue 4: C-Rule Update Performance

Issue 4 focuses on the performance of updating a C-Rule in a C-Rule based system. How long does it take to update the C-Rule base and how often is it required? Can this updating be finished during the interval between network packets in order for the system to maintain detection at wire speed?

Since the C-Rule method is proposed to improve expert system's performance, updating a C-Rule cannot be time consuming since that would slow down the expert system. To address this issue, we focus on the three parameters:

- Number of updates of C-Rule: number of the C-Rule updates happening during detection
- Number of C-Rule generation per update: Number of C-Rules generated during each C-Rule update
- C-Rule update time per update: the time to finish all C-Rule generation for one C-Rule update

We have categorized the multi-packet attacks into three categories in order to analyze these three parameters. In Section 5.4.4, we explain in general how to update CAM using CAM writing operation and what these 3 parameters are for CAIPPES. In Section 5.6.4, Section 5.7.4, and Section 5.8.4, we explain in detail for each category what kind of new C-Rules are going to be updated. In Section 5.6.7, Section 5.7.7, and Section 5.8.7, statistics from the experiments show for each category how many C-Rule updates happen in the system and how many C-Rules are updated in every update, as well as how many clock cycles the updating would take. These statistics will also show whether the C-Rule updating can keep up with wire speed detection for Gigabit networks.

Issue 5: C-Rule Scalability

Issue 5 focuses on the C-Rule scalability problem. How many C-Rules does the system have? Can these C-Rules all fit into one or multiple commercially available CAMs? If there are a huge number of C-Rules, it is not feasible to fit them into CAMs' depth and we are thus unable to exploit O(1) match complexity. Therefore, this question concerns the feasibility of using a CAM to accelerate the detection.

To answer these questions, we focus on the parameter "total number of C-Rules in CAM" to find out how many C-Rules are in CAM from time to time. We have categorized the multi-packet attacks into three categories. In Section 5.4.5, we explain in general what affects the number of C-Rules in the system. In Section 5.6.7, Section 5.7.7, and Section 5.8.7, statistics from the experiments show for each category how many C-Rules the system has at one time for each detection, including analysis of these numbers. These statistics will also show whether or not the average number of C-Rules in a system can be comparable to a CAM's depth.

Issue 6: Experimental Validation

Issue 6 focuses on the experiment and validation for the C-Rule method in real world traffic to see how the system performs for real traffic. A prototype system and input simulation data are required for the experiment. Performance analysis from the simulation results will address the above issues.

In Section 5.4.6, we show the details of the prototype system and simulation data. A list of performance analysis parameters to be examined is provided as well. In Section 5.6.7, Section 5.7.7, and Section 5.8.7, performance analysis for each category is presented.

5.4.1 Mapping C-Rule's Condition Evaluation into a CAM

C-Rule's condition consists of two criteria information as shown in Figure 5.11: packet criteria and the inter-factual criteria between packet facts and context facts. The inter-factual criteria are criteria on the packet fields based on different network contexts.

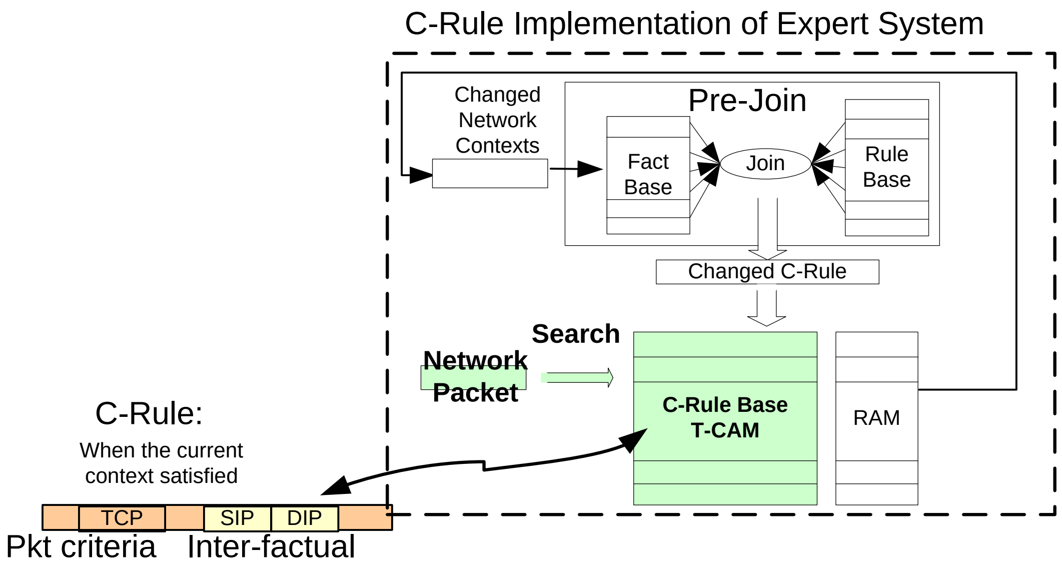


Figure 5.11 Packet Searches C-Rule Base

There are three types of evaluation required for a C-Rule condition:

1. Exact matching with a specific value in a field

2. Masked matching with some “don't care” bits in the value in a field
3. Range matching in a field
4. String matching in payload

Requirements 1 and 2 can be easily implemented in a Ternary CAM. Requirement 3 has been solved in Chapter 3 using a novel encoding scheme in Ternary CAM. Requirement 4 has been solved in Chapter 5 except for complicated strings in Perl Regular Expression. In our experiment, no Perl regular expressions are required

Therefore, the matching ability of a Ternary CAM can handle the requirements of C-Rule condition evaluation except for Perl regular expression in payload matching. C-Rule condition evaluation can be implemented in a Ternary CAM in the same way as Snort Rules in Chapter 4. The inter-factual criteria will be mapped to binary values in the packet fields. Therefore C-rules requires the same format as Snort in Chapter 4, as well as the same width of CAM.

5.4.2 Mapping C-Rule’s Action

CAM is applied to the C-Rule method to hold C-Rule’s condition part. We can use a RAM to hold opcode for C-Rule’s action part. When CAM matches at a particular address, this address becomes the input to the RAM. Thus the output of the RAM contains an opcode for the matched C-Rule’s action. Therefore any action can be implemented in this CAM plus RAM architecture.

There are two kinds of actions in the system. One is to send output results. In our experiments, we simplified the actions to be only “pass”, “drop”, and “log”. The output also includes a warning message if needed for that packet to report the attack types.

Another action is to modify the expert system internal context fact or C-Rules. Modifying context facts can be handled by the pre-join block, which is explained in Section 5.5.5, Section 5.6.5, and Section 5.7.5 respectively for each category. Updating, adding, or modifying C-Rules can be implemented by the writing and invalidating operations in the CAM. These two are explicitly addressed in Sections 5.4.4.

5.4.3 C-Rule-based Architecture Latency and Execution Time

Latency time is defined as the time difference between when a system receives a network packet and when the system generates the intrusion detection result. When the detection result is obtained, the system can decide what to do with the packet, whether to pass, drop or alert the user. The latency time of CAIPES is important for many real time network applications.

In a conventional expert system, the latency is the time to do the joining between all facts and all rules match plus the action time. The matching time is proportional to the matching complexity O(C*R).

As shown in Figure 5.12, conventional expert system latency T1 is the sum of Matching time plus action time:

T1 = T(matching) + T(action) = *k * O(C*R)* + T(action)

(*k* is one search operation time in expert system database)

From Section 5.2.2, this latency time will be longer than 100,100 ns using current processors.

However, in the Contextual Rule based method, there are two explicit parts of matching: Pre-join matching and C-rule matching. The incoming packet will be compared to C-rules instantly to get the intrusion detection result. Therefore, latency of CAIPES is the C-rule matching CAM operation latency. Pre-join matching will then join any new context resulting from this match to all rules, thus preparing the new C-rule base for future packets. In order for the system to process at wire speed for gigabit networks, we make the assumption that the pre-join process and the activation C-Rule process can be finished before the next packet arrives.

As shown in Figure 5.12, CAIPES expert system latency T2 is the sum of matching time of the C-Rule base (a CAM) plus action time (RAM read time):

T2 = T(C-Rule matching) + T(action)
= T (match in CAM)* O(1) +T(RAM)
= 8 clock cycles + 2 clock cycles
= 10 clock cycles
= 100 ns for IDT 75K series CAM

Pre-join and activation are performed after the system output is sent. These operations need to be completed before the next packet is arrives. Their execution time doesn't count in the latency time.

In order for the system to process at wire speed for gigabit networks, we assumpe that the pre-join process and the activation C-Rule process can be finished before the next packet arrives.

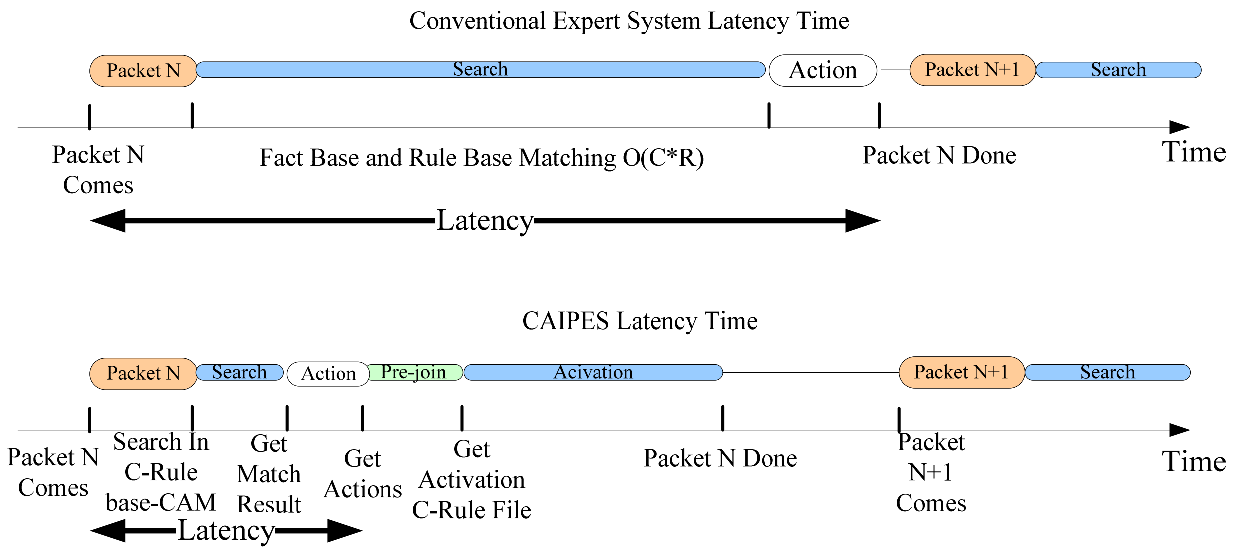


Figure 5.12 Latency Comparison Between Conventional Expert System and CAIPES

Execution time in a conventional expert system is the same as the latency time. As shown in Figure 5.13, conventional expert system execution T3 is the sum of Matching time plus action time:

T3 = T1 =T(matching) + T(action) = k * $O(C*R)$ + T(action)

(k is one search operation time in expert system database

From Section 5.2.2, this execution time will be longer than 100,100 ns using current processors.

As shown in Figure 5.13, CAIPES expert system execution time T4 in is the latency time plus the Pre-join and Activation time if there is a match from packet searching the C-Rule base.

T4 = T(C-Rule matching)

+ T(action)

+ T(Pre-joining contexts with rules)

+ T(modification CAM)

= T (match in CAM)* O(1) +T(RAM)

+ T(Pre-joining contexts with rules)

+ T(modification CAM)

= 10 clock cycles + T(Pre-joining contexts with rules) + T(modification CAM)

There are even more times when there is no match from packet searching the C-Rule base (in our experiments only 10% of packet searches result in matches in the C-Rule base, even under network attack). In this case, CAIPES' execution time equals to its latency time and is thus even faster as shown in Figure 5.13.

T4 = T(packet searches in all C-Rules) = 8 clock cycles = 80 ns

In order for the system to process at wire speed for gigabit network, we assume that the pre-join process and the activation C-Rule process can be finished before the next packet comes.

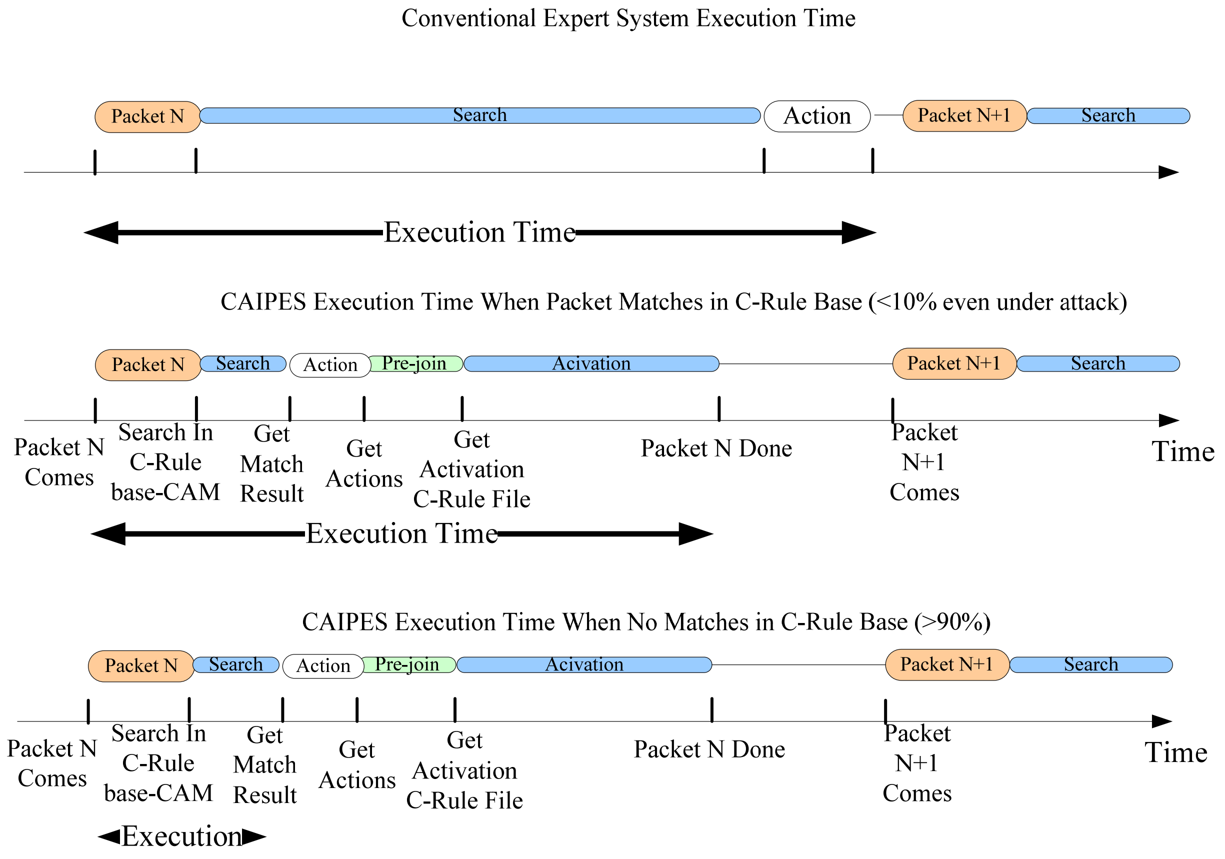


Figure 5.13 Execution Time Comparison Between Conventional Expert System and CAIPES

The acceleration of CAIPES lies in the parallel search engine CAM's fast speed of C-Rule search and its fast update speed. It also lies in the saving of unnecessary searches of facts with rules if no matches are found in the packet searching C-Rule base.

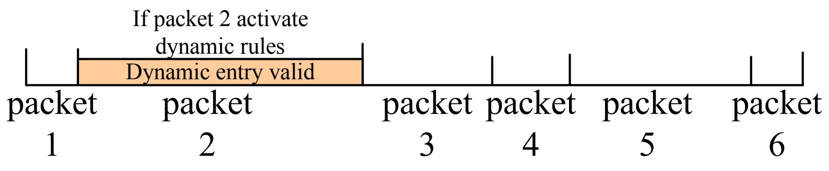
CAIPES architecture also has the potential to exploit the pipeline ability in CAM search to accelerate the system output. Extra hardware and control mechanism is needed to use the pipeline search operation.

5.4.4 C-Rule Update Performance

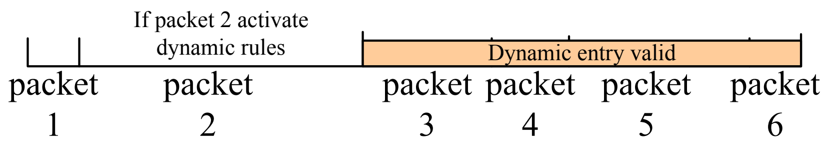
Dynamic C-Rule generation is the most important functions in CAIPES. There are three steps involved in this process:

- Step 1: A new network context fact gets changed. It is tagged with related fact, related rule, and activating fields if available, and sent to the Pre-join block.
- Step 2 (Optional): The Pre-join block joins the new facts and related rules to generate the C-Rules.
- Step 3: C-Rules are added or invalidated in the C-Rule base (a Ternary CAM) through CAM writing operations and invalidating operations.

The dynamic activation of a C-rule is different from dynamic activation in single-packet attack detection, as shown in Figure 5.14. Single-packet detection activates rules only during the same packet time as Figure (a) illustrates; Multi-packet detection activates rules only after this packet as Figure (b) illustrates.



(a) Dynamic Activation in Single-packet Detection



(b) C-Rule Activation in Multi-Packet Detection

Figure 5.14 Difference Between Single-Packet Detection and Multi-Packet Detection

In a Gigabit network, we have a minimum 500ns time interval between two packets. If the updating can be done during this 500ns, we have no problem to keep up with the traffic.

Activation of C-Rule in CAM

The new Contextual Rule will be added into the C-Rule base, a Ternary CAM. Pre-join generates a proper CAM address for the C-Rule to be written in the CAM. This address is also tagged to its matched fact in the fact base.

The CAM will be configured as 576 bits wide. As shown in Figure 5.15, we allocate a specific segment in the CAM for each attack. The packet criteria rule will be mapped to CAM using a format similar to Chapter 4 single-packet attack rules.

At the bottom of the segment is the first C-Rule with packet parameter (inter-factual criteria) specified as “don't care”. This C-Rule always stays in the CAM. Above it, all entries are pre-filled with similar rules, except that in the inter-factual criteria fields Mask bits are cleared so that these data bits are no longer “don't care” bits as in the bottom C-Rule 1. Also, these pre-filled rules are set as “not valid” in the beginning. A 576-bit search command containing the packet information is to be compared against these rules.

When new C-Rules are generated from the Pre-join block, CAM will use 72-bit “write” command (which is the only write command in CAM) to activate one “invalid” rule nearest to the bottom. The write command will also write to the specific inter-factual criteria to the data array. Since the mask array is pre-filled, the worst case is to write 8 times for the data array if the parameters spread to all 8 parts of the 576 bits. This costs 9 clock cycles for burst writing, and in current IDT CAM technology [85], it is 90ns. In case this C-Rule needs to be overwritten by a new C-Rule which has different mask bits, the mask array needs to be modified too. Therefore, the worst case costs 9 * 2 =18 clock cycles and 180ns. This time is much smaller than the 512ns minimum interval time between two back to back packets in Gigabit network. We will see from our experiments, the real writing cycle number per C-Rule is much smaller than the worst case of 18 clock cycles. It is an average of 2-6 clock cycles.

In the event that a lot of different attacks happen each with different parameters, the pre-allocated space may be used up. The Pre-join block will manage the CAM space and propose an empty entry in a “free for use” segment, as shown in Figure 5.16. Entries in “free for use” are not pre-filled with any mask array data or data array data. Therefore, adding a new C-Rule there costs 9 clock cycles for burst writing of data array and another 9 clock cycles for mask array. In total, it is 180 ns in current IDT CAM technology [85].

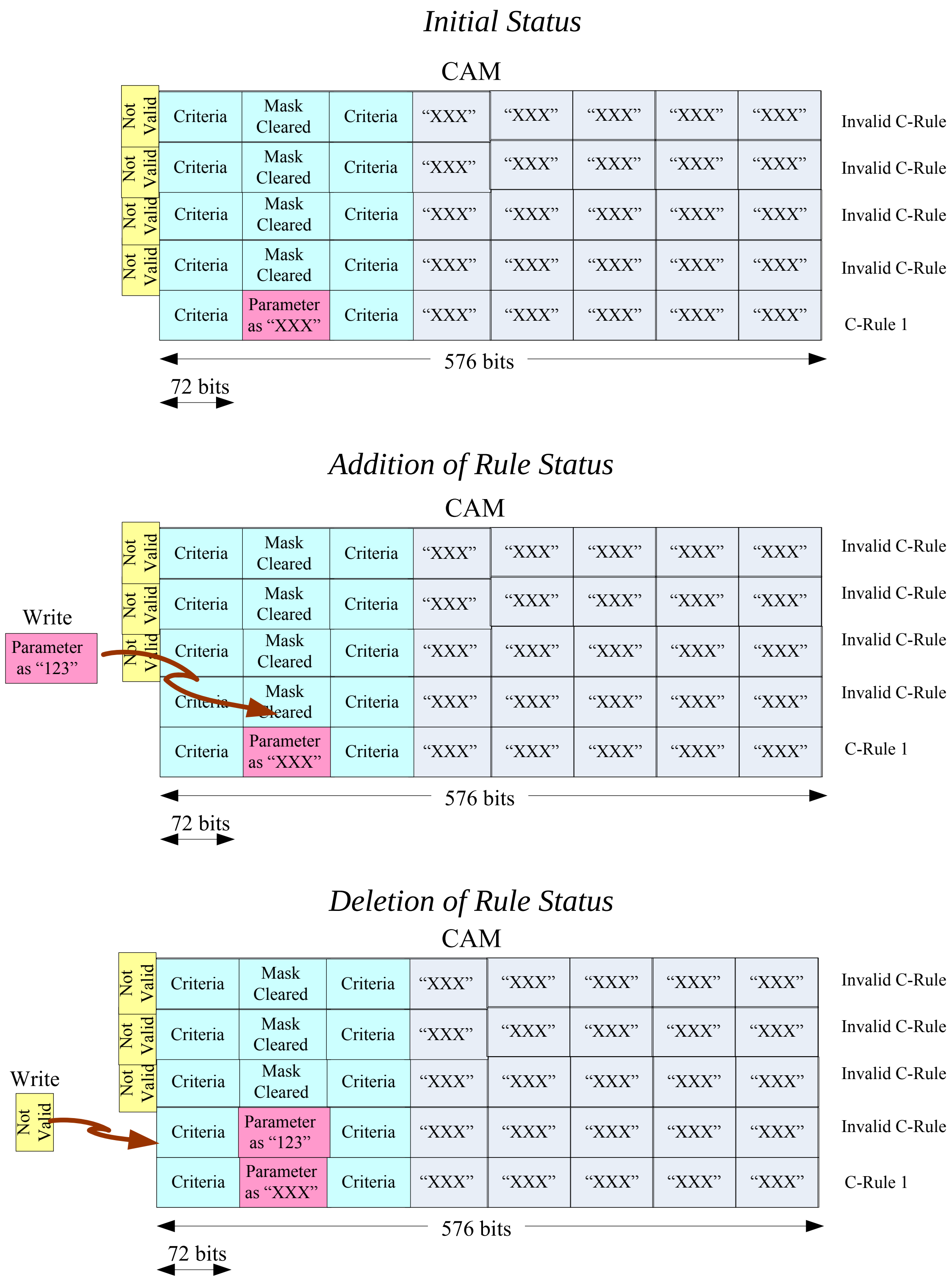


Figure 5.15 Activation and Invalidation of C-Rules in CAM

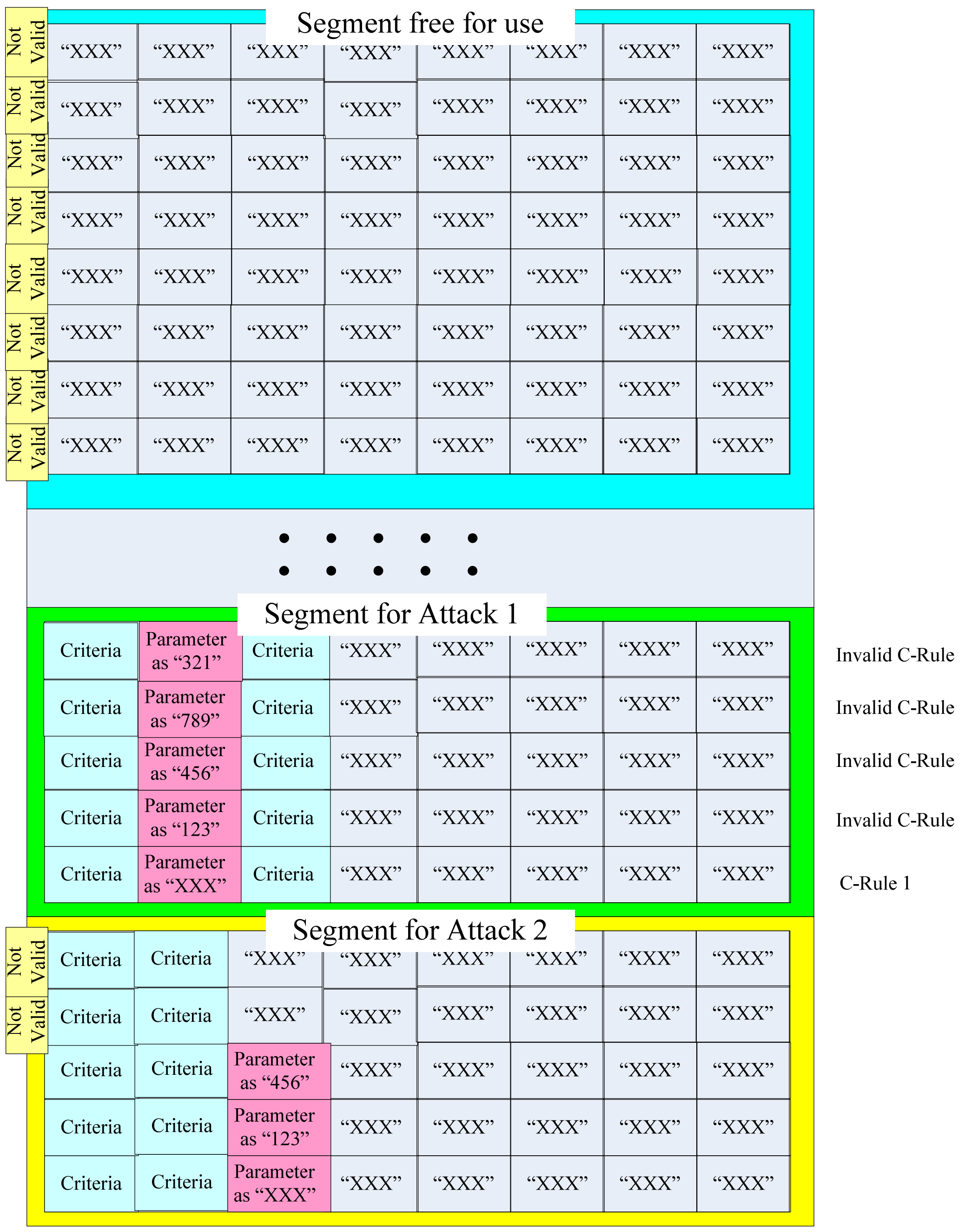


Figure 5.16 CAM Segment Allocation for Multiple Attacks

Invalidating C-Rules in CAM

Invalidating C-Rules happens when some C-Rules become outdated after some time or invalid because of some other C-Rules. Usually the pre-join or RAM block will issue the invalidating command.

The process of deleting one C-rule occurs as follows: we issue the *invalidate* command to the CAM with the obsolete data entry content. The CAM will find all matches to this data entry and invalidate them by setting their valid bits as ‘0’s. This can be done in one clock cycle. If we have multiple inter-factual criteria fields in one C-Rule, we only need to invalidate the first of them because doing so already invalidates the 576 bits entry as one rule. This deletion takes one clock cycle, and is 10ns in current IDT CAM technology [85]. This invalid entry empties the space and can be used when a new C-rule need to be added. The specific data in the parameter fields doesn't need to be cleaned during the deletion because it will be over-written anyway next time for the new C-Rule.

Deletion of one rule in the “free for use” segment still costs only one clock cycle which is 10 ns in current IDT CAM technology [85].

Number of Updates for C-Rules

A C-Rule in the C-Rule base is a combination of current context facts and rules. A C-Rule update happens when a new context fact matches one or more rules in rule base. Therefore the number of C-Rule updates is the number of new context facts that were ever generated during detection, which is far less than the product of context fact number and rule number:

C-Rule_Update_number = SUM (Context) = O(Context) << O(Context*Rule)

Number of C-Rule Generation Per Update

Since not every current context can be related to all rules and joined into a C-Rule, the amount of C-Rules generated per updates is less than O(Context*Rule). Instead, it is the number of matches that a pre-join block could find for the current context facts:

C-Rule_Generation_perUpdate = 1 Context * Related_rules = O(Related_rules) << O(Context*Rule)

Related rules for one context are usually limited. In category 2 detection, it is the types of statistics tracking detections a context fact would be involved. In category 3 and 4 detection, it is the possible transition from one state. Most cases, the related rule number is less than 10.

C-Rule Update Time per Update

The time to finish one updates is the product of generated C-rules number and activation time per rule in CAM:

C-Rule_perUpdate_time = O(Related_rules) * Required 9B_CAM_entries_segment * CAM_update_time

If the update time is smaller than 512ns which is the minimum packet interval between two back-to-back packets in a Gigabit network, the C-Rule method can keep up with line-speed Gigabit detection.

5.4.5 C-Rule Scalability

Total Number of C-Rules Generated During Detection

A C-Rule in the C-Rule base is a combination of current context facts and rules. Let us look at the total number of C-Rule generated during detection. Since not every current context can be related to all rules and joined into a C-Rule, the amount of C-Rules generated during detection is less than O(Context*Rule). Instead, it is the number of matches that a pre-join block could find between the existing context facts and rules:

C-Rules number = SUM (1 Context * Related_rules) = Context * Average(Related_rules)

= O(Context * Related_rules) << O(Context*Rule)

Two variables determine the C-Rule update frequency: (1) the amount of different related context in the traffic which can trigger the rules; (2) the related rules to each context. They depend on the following factors:

1. How special the packet criteria are and how often the matched packet will appear in the traffic. If the packet is normal and happens very frequently, then the C-Rule base will be frequently matched and modified. For example, usually < 0.1% packets in the traffic are ICMP, but almost 90% are TCP packets, therefore a TCP rule will generate more C-rules than a ICMP rule when detecting the same traffic because apparently there are more TCP contexts in the traffic. Assume the percentage of this traffic is P_{traffic}.
2. How different the related contexts are from each other. Every unique context will generate a totally new rule. Therefore, a context with more fields of packet information has more chance to be different from others. For example, connection based contexts examine destination and source IP address and port number. It has more different

contexts than source IP alone based contexts in the same traffic, thus more C-rules because of more contexts. Assume the percentage of different context is P_{diff}.

3. How many rules will be activated for one context. This depends on the detection rules.

Therefore total C-rule generation = packet_number * P_{traffic} * P_{diff} * Related_rules

Total Number of C-Rules in CAM

The C-Rule number in CAM is actually much smaller than O(Context * Rule) because not every context can be joined with all rules. A context can only be joined with it related rules.

In addition to the factors that affect the number of C-Rule generation, the C-Rule number in the CAM also depends on three other factors:

1. The timeout window which would age the rules and invalidate the outdated ones;

2. The new state transition rules which will overwrite the old state transition rule in Category 3 and Category 4 detection.

3. Those state transitions that reach the final state and would invalidate the related rules in Category 3 and 4 detection.

Therefore, for category 2 detection, the number of C-Rules in a CAM is number of current contexts number existing in the time window, multiplied by the types of traffic tracking.

C-rule = packet_number_in_window * P_{traffic} * P_{diff} * traffic_tracking_types

For Category 3 detection, the number of C-Rules in a CAM is number of possible transitions from the current intermediate state contexts.

C-rule = intermediate_state_contexts *possible transitions

For Category 4 detection, it is the product of these two. We analyze these two scalability issues in more depth with real data for each category, which are covered in 5.5.7, 5.6.7, and 5.7.7.

5.4.6 Experimental Validation

A prototype of this CAIPES is shown in Figure 5.17. Sniffing data from the Lincoln Lab dataset is reformatted through a software protocol analyzer to fit the format of Rules in CAM. The driver is a System C controller for the C-Rule base to search the CAM, activate new rules or invalidate new rules in the CAM. The CAIPES C-Rule base is modeled by a System C behavior model, which will output the Match result to a result file. The Pre-join block written in Perl will read the matched result and calculate the C-Rule change for the C_Rule base. These C-Rule activation and invalidation are written into a file which will be read by the driver.

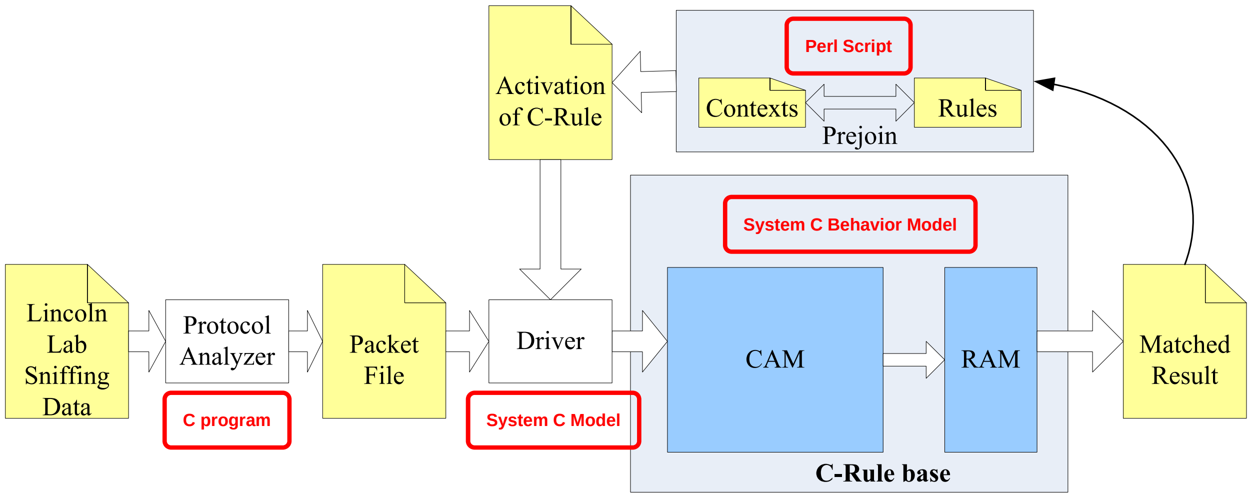


Figure 5.17 Prototype System for CAIPES

The details of the driver's execution is shown in Figure 5.18: the driver will read packets from the packet file according to their time stamp, and sends them one by one for a search in the C-Rule base. During the interval between two packets, the driver will wait and see if the first packet finds a match in the C-Rule base. If there is no match, the driver will wait till the second packet comes when its timestamp is reached, and then send it to search the C-Rule base. If there is a match between the first packet and the C-Rule base which will activate a new C-Rule, the driver will read the activation information from the "activation of C-Rule" file generated by the Pre-join block, and perform the proper operation to the CAM. When this modification time is smaller than the packet interval time, we can process the packet in line speed. When this modification time is greater than the packet interval time, we will hold the packet in a fifo and wait until the modification is done before we perform a search in the C-Rule base.

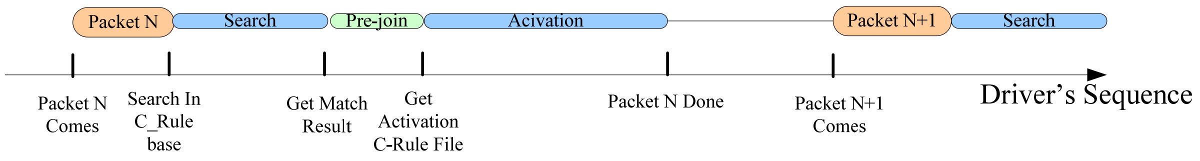


Figure 5.18 Driver's Sequence of Actions to Drive CAIPES

For simulation of the prototype system, we use Lincoln Lab Data sniffing file as the input packet source. Three attacks each from Category 2, Category 3 and Category 4 are chosen to run through the prototype. In order to analyze the performance of the method, we choose 2 packets files to run the simulation on each attack. One of the packet files is an attack-free file containing normal clean packets to simulate the CAIPES performance in normal network environment. The other packet file contains the specific attack we are going to detect. This simulates the CAIPES performance when an attack takes place.

We analyzed system performance using a lot of parameters. In order to see monitor how CAM is used in the system, there are CAM search and match parameters to measure the number of search operations happen during detection, the number of matches in the C-Rule base, and the number of matched newly activated C-Rules in CAM. To see the C-Rule updating performance, there are C-Rule activation and invalidation parameters to measure the total number of activation times during detection, the total number of activated C-Rules, the total number of clock cycles of CAM writing and the total number of invalidation. To see C-Rule scalability, there is C-Rule number parameter to check the maximum number of C-Rules in C-Rule base that are existing at a point of time. There is also a system speed parameter to check number of slower updates where the next packets arrives before the C-Rule update is done. We use 200ns as the interval between the last search of the previous packet and the first search of the next packet. This equals to the fastest transmission speed in more than two gigabit/sec networks.

The input data is accelerated Lincoln Lab data. It is originally 100Mbps and we accelerated them to be 100 times faster. In order to further accelerate the simulation, we replace the interval between two packets with a new accelerated interval time, which is the same time when the original interval time is less than 200 ns, and 200ns when the original interval is beyond 200ns. This equals to the fastest transmission speed in more than two gigabit/sec networks. From these results we can see C-Rule method can keep up with this traffic.

5.5 DETECTION OF CATEGORY 1 NETWORK INTRUSIONS

Category 1 intrusion is single-packet attack. To detect Category 1 attack, the IDS should examine each incoming packet, and check the header information and payload string to see if the current packet matches any signature of attacks in Category 1. This has been solved in the same method as in Chapter 4.

Category 1 detection depends only on current packet information. No network contexts are involved. Only the packet fact but no context facts are used in the detection. Therefore, no pre-join functions are required and the C-Rules in Category 1 are the same as original rules. The only functions required for Category 1 detection is packet searching C-Rule base, as well as system output functions. This can be implemented by a CAM and a RAM in a similar way as that described in Chapter 4.

5.6 DETECTION OF CATEGORY 2 NETWORK INTRUSIONS

Category 2 attack is multi-packet attack containing flooding of packets. This flooding of packet is malicious and do harm to networks and computers.

Problem: A type of multi-packet attack is a flooding of packets that if they are only analyzed individually, appear to be legitimate traffic. Thus, it is the frequency of the packet's occurrence, rather than the individual packet, that leads to intrusions. They include network bandwidth exhaustion, information exposition, and other legal but abusive traffic. An analysis of the Lincoln Lab IDS evaluation dataset shows that 29% of multi-packet attacks in are Category 2 attacks. These attacks are enumerated in Table 5.1.

Example: An example of Category 2 attack is the Smurf attack as shown in Figure 5.19: Smurf attack is a network bandwidth exhausting DOS attack. It occurs when an attacker outside the network sends a number of ICMP echo packets to the broadcast address of a LAN with a spoofed insider source IP. Each computer on the LAN replies the ping requests to the victim whose IP has been spoofed by the attacker. Network will be flooded with all the replying packets which exhaust the bandwidth. The number of ICMP replying packets decides how severe the bandwidth exhaustion would be.

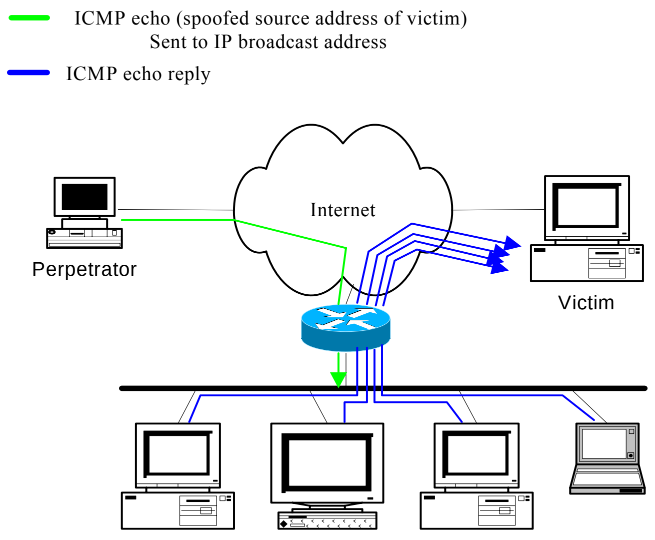

ICMP echo (spoofed source address of victim)
Sent to IP broadcast address
ICMP echo reply
Internet
Perpetrator
Victim

Figure 5.19 Attacking Mechanism of Category 2 Smurf Attack

Detection: To detect category 2 attacks, an expert system analyzes multiple packets and their occurrence statistics. It detects malicious traffic in category 2 by recognizing the frequency of a packet's occurrence rather than the internal pattern of a packet or sequence of packets. It is able to detect flooding type attacks, where a high frequency of a packet's occurrence indicates an attack; or detects abuse of a legitimate packet deviating from normal traffic statistics. This detection can also determine the severity of an intentional attack according to the number of irregular packets.

Supported detection includes:

- **Profile-based anomaly detection:** Collecting statistics concerning features of attack-free traffic (e.g., bandwidth and connection duration) to make a profile for each host. The profile is updated later on. This detects behavior that deviates from the profile of standard traffic.
- **Threshold anomaly detection:** Setting a threshold value for the occurrence frequency of specific legitimate packets. Any traffic that goes beyond the threshold is considered malicious traffic. This method is capable of detecting flooding attacks.

- **Attack severity identification:** Setting threshold values for different severity levels according to different occurrence frequencies of specific packets.

5.6.1 Expert System Functions Required For Category 2 Detection

To recognize a Category 2 attack, expert system rules describing it check the statistics of a specific kind of packet during a period of time. The statistics number will show the severity of the attack. The context fact in Category 2 detection is *"Statistics fact"* where the timestamp of the event occurrence is included. The rule condition in Category 2 has *"Statistics Criteria"* as its context criteria to check on the statistics facts.

The functions required to detect Category 2 attacks are listed in the order of how a flooding attack is detected as follows:

1. **Packet Searching C-Rule Base:** Packet Searches C-Rules, finds that it has the specific packet pattern of the flooding packet
2. **Modifying Statistics Facts:** Generate the changed statistics fact
3. **Pre-joining in Pre-join Block for Statistics Checking:** Adding a new time stamp of the event; Moving time window and Updating Statistics; Checking statistics value with threshold in rules;
4. **Activating New C-Rule:** Activating a new C-Rule with the flooding's context such as IP address or port number information embedded in the C-Rule
5. **Repeat 1-2** when more packets in the flooding come and match the newly activated C-Rule. No new C-Rule will be activated if the statistics value hasn't reached "threshold-1" number in Step 2. Once the statistics value reaches "threshold-1" value, go to Step 5.
6. **Activating New C-Rule:** Modify the C-Rule activated in Step 3 to a new C-Rule with the same flood's context in its condition, but an additional alert bit in its action.
7. **Packet Searching C-Rule Base:** Packet searches C-Rule, finding one more flooding packets matches the newly modified C-Rule. This means the flooding's threshold has been reached
8. **Dropping Packet and Sending System Alert:** Drop any more flooding packets and send an alert message to the system.

Therefore, five functions are required in Category 2 attacks. They are "Searching C-Rule base", "Modifying Statistics Facts", "Pre-joining", "Activating C-Rules" and "Dropping Packets and Sending System Alert". Details of these functions are as follows:

5.6.2 Category 2 Function 1: Searching C-Rule base

This function is the basic function of C-Rule method. It is the same for all three types of multi-packet attacks. This function has been described in Section 5.4.1.

5.6.3 Category 2 Function 2: Modifying Statistics Facts

Changed network statistics fact is generated from a matched C-rule. Each changed network statistics fact can be seen as an event that will lead to traffic flooding. As shown in Figure 5.20, it includes information of the attack ID, the parameters to identify the attack incident context, the timestamp, the related rule number and related fact number.

This fact will be sent to the pre-join block to be stored and evaluated there. The following Function 3 illustrates the process.

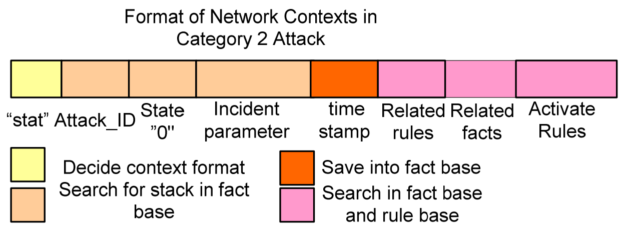


Figure 5.20 Format of Statistics Fact in Category 2 Detection

5.6.4 Category 2 Function 3: Pre-joining for Statistics Check

Figure 5.21 shows the details of this pre-joining. First, the changed statistics fact goes into the Pre-join block by adding a new time stamp of the event. It is stored in the fact base according to the "related fact" field. The counter updates with an increment. Then the pre-join block shifts the time window, drops the outdated events from the window, decrease the counter to keep the current statistics for the new window. Finally, it checks the statistics value against the threshold value in the rules and finds any satisfied Rules to generate new C-Rules.

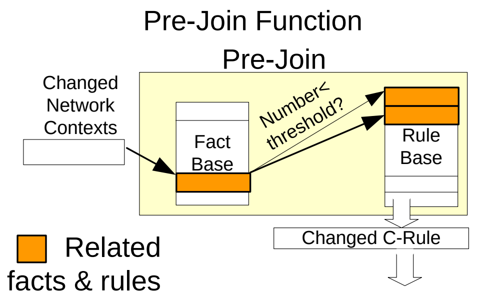


Figure 5.21 Pre-join Block in Category 2 Detection

Function 3.1: Adding a New time Stamp and Counter increment

The Fact base in the Pre-join block saves changed statistics fact in a format shown in Figure 5.22. Context facts with the same attack ID and same parameters but different timestamps are combined in one stack. The related time window information for that stack is extracted from rules and is saved in the stack, for simpler execution of Pre-joining between the rule base and the fact base. The related C-Rule address is also stored for later fast deletion of the C-Rule in C-Rule base.

When a new network context comes in, the Pre-join function will first read its “related fact number” field, and find the stack with the same attack ID and parameters. If such a stack exists, the timestamp of the new event will be added into the stack. Time window information in that stack will be then used to shift the window in the stack in the next step. The number of valid timestamps in the time window will be checked by the next step as well.

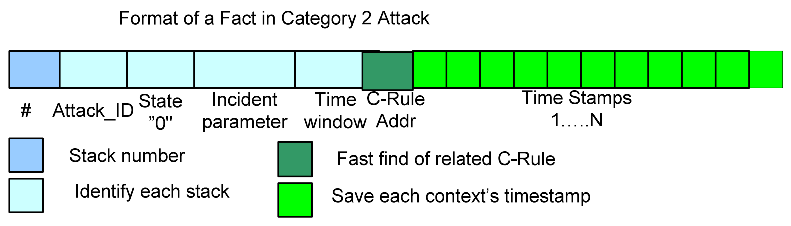


Figure 5.22 Format of Facts Stored in Fact Base in Category 2 Detection

Function 3.2: Time Window updating

Category 2 detection needs to check the frequency of the occurrence instead of just an occurrence number. This requires timing information with the counter. We use a user-defined time window size and count the occurrence number inside this time window in order to calculate the frequency. To track the frequency continuously and avoid sampling gap, we use a time window that slides as time progresses.

In a brute-force algorithm, we will use each occurrence event to trigger a new time window of tracking statistics, to avoid sampling gap. In Figure 5.23, the X axis shows time and events occurring at different times. For an example, we choose a window size of 10 time units with an attack threshold of 6 attacks inside any window.

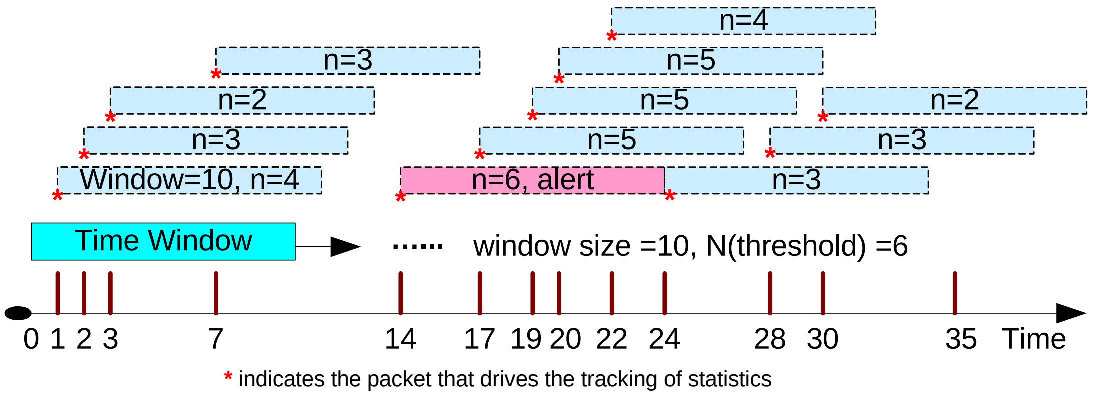


Figure 5.23 Conventional Time Windows Change as Time Progresses in Category 2 Detection

Every event occurred and displayed on the X-Axis will trigger a new time window. Each time window will collect the number of events happening during its tenure. When time passes the window time unit size, the window will expire. When the attack threshold is reached during any window tenure, the system will send an alert. The red stars in Figure 5.18 indicated the events that trigger a new window. The blue windows are those time windows where the threshold is not reached. The pink window is the window where the threshold is reached. In Figure 5.23, there are 6 occurrences from time unit 14 to 24, which is the only time window which generates an attack alert. We can see from Figure 5.18 that multiple time windows are processing and calculating at a point in time.

Each time window is a network context. A large number of time windows at a point of time leads to a large amount of network contexts at that time. This means a lot of calculations in the Pre-join block where network context and rules are joined. It also leads to frequent generation and updates of C-Rules. In order to improve the C-Rule efficiency, reducing unnecessary time windows is desired.

An optimized algorithm of tracking occurrence statistics is shown in Figure 5.24. Instead of every occurrence event triggering a new window, in this algorithm only necessary new time windows are triggered. In Figure 5.24, those windows shown in blue or pink are necessary time windows while the white ones enclosed in dashed lines are unnecessary time windows that won't be started at all.

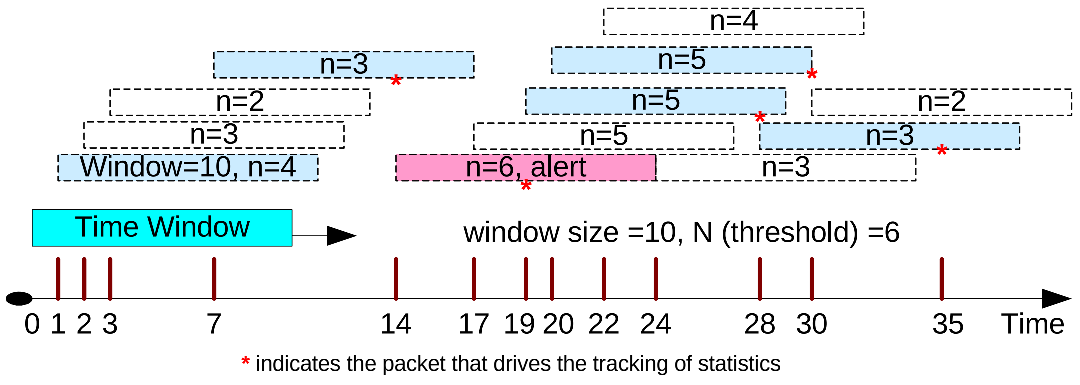

n=4
n=3
n=5
n=2
n=5
n=2
n=3
n=5
n=3
Window=10, n=4
n=6, alert
n=3
Time Window
window size =10, N (threshold) =6
0 1 2 3 7 14 17 19 20 22 24 28 30 35 Time
* indicates the packet that drives the tracking of statistics

Figure 5.24 Novel Time Window to Reduce Unnecessary Windows in Category 2 Detection

A time window slides when time progresses. Old events leave the time window and new events join the time window. If the statistics in a time window doesn't reach the threshold, then old events leaving the time window won't make the statistics any higher to reach the threshold. Time windows with old events leaving but without new events joining can be characterized as unnecessary time windows. Only a time window with new events joining it has the possibility of exceeding previous statistics in older time windows.

For example, if we consider to the time window started at time 1 and the time window started at time 2, the following comparison can be made. The time window started at time 2 loses the event happened at time 1, without any new events joining it compared to the time window started at time 1. Therefore, its statistics can only be lower (3 occurrences) than the time window started at time 1 (4 occurrences). Thus the time windows started at time 2 and 3 are both unnecessary.

However, an event happening at time 14 added a new event which is not in the time window from time 1 to time 11. Therefore this event triggers a new necessary time window. The first new necessary window including this event is the time window started at time 7. We can find this window by first subtracting the window size of 10 from the event time of 14 and getting result time 4, which means windows after time 4 will include the new event. Then we find the event which happened at time 7 is the first event after time 4. Therefore, the time window started at time 7 is a necessary window for us to track and it was triggered by an event happening at time 14.

The next necessary time window is the time window started at time 14. The event happening at time 19 is the first event after time window from time 7 to 17. This event triggered a new time window started at time 14 (the first event after time 9). In this new time window, the statistics reaches the threshold and sends an alert to the system.

With the optimized sliding time window calculation, it is necessary to store every event's occurrence timestamp in order to avoid recounting for a new time window, as shown in Figure 5.25. To track and count statistics of a specific type of traffic with same parameters such as source address and destination address, it only needs one shift register to store all its events. To track different time windows, we only need to define the different start times of those time windows, which can be done by writing the time value into "start_time" registers. The number of registers in use is the same as the concurrent time windows at a point in time.

As time progresses and events added into the "timestamp" shift register, old time windows expire and new time windows appear. Events that are older than current time minus window size are expired forever. Therefore they can be cleared from the shift register and also the start_time register meaning this time window has expired. Remaining events will be counted in the new time window if there are any. The first valid index in the "timestamp" shift register from left to right simply indicates how many valid events there are in the new window currently. In the optimized algorithm, clearing the old events and counting for the new window are triggered when new events are added. Figure 5.26 illustrates the detailed steps of how an optimized time window works mentioned above.

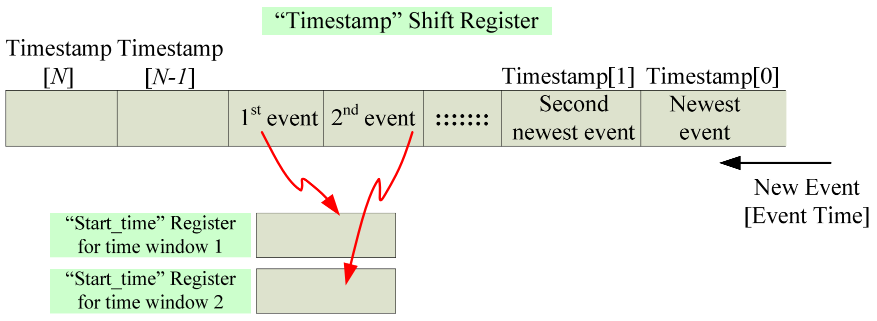
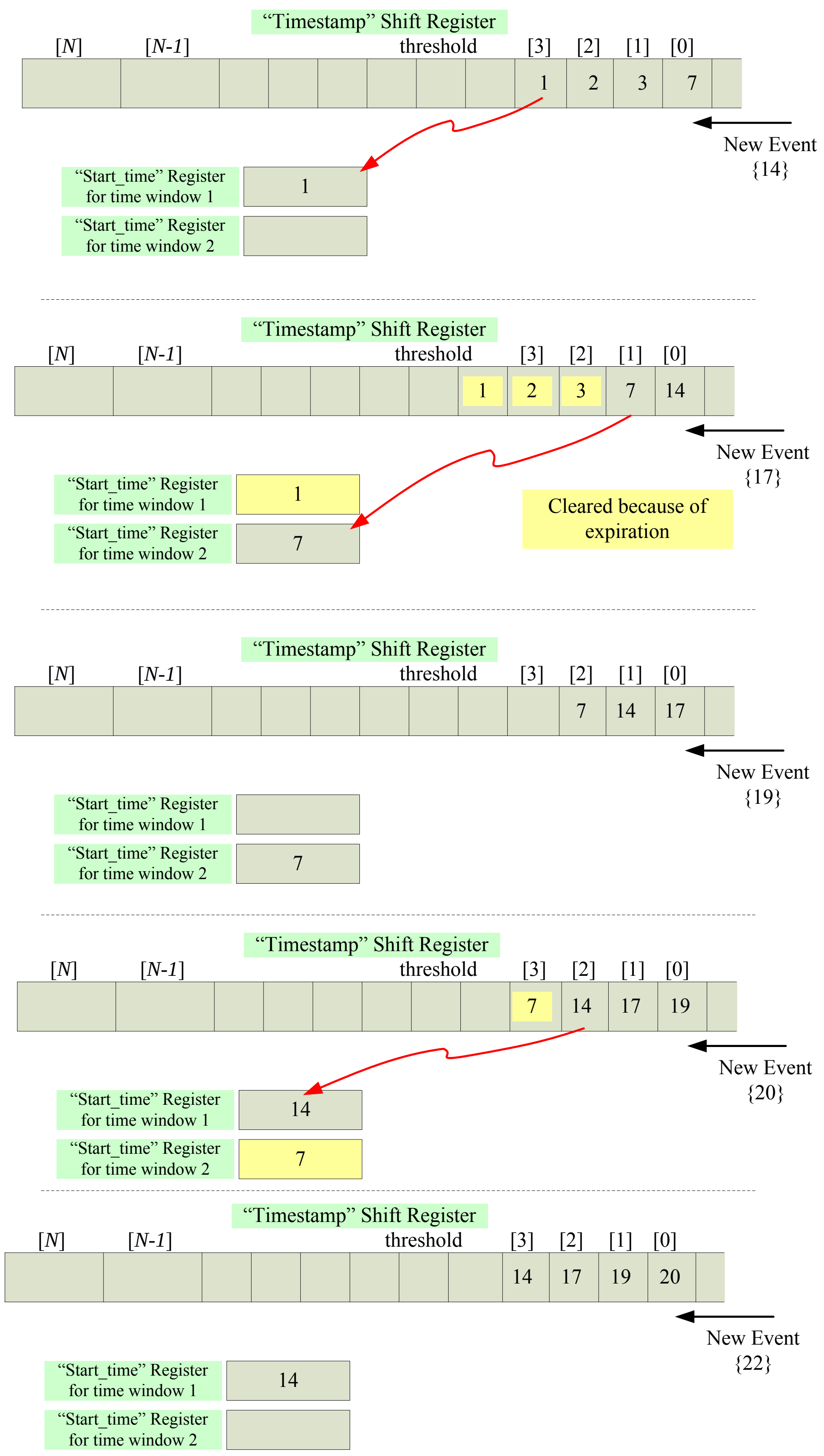


Figure 5.25 Timestamp Stack in Fact Base in Category 2 Detection


"Timestamp" Shift Register
[N] [N-1] threshold [3] [2] [1] [0]
1 2 3 7
New Event {14}
"Start_time" Register for time window 1
1
"Start_time" Register for time window 2
"Timestamp" Shift Register
[N] [N-1] threshold [3] [2] [1] [0]
1 2 3 7 14
New Event {17}
"Start_time" Register for time window 1
1
"Start_time" Register for time window 2
7
Cleared because of expiration
"Timestamp" Shift Register
[N] [N-1] threshold [3] [2] [1] [0]
7 14 17
New Event {19}
"Start_time" Register for time window 1
"Start_time" Register for time window 2
7
"Timestamp" Shift Register
[N] [N-1] threshold [3] [2] [1] [0]
7 14 17 19
New Event {20}
"Start_time" Register for time window 1
14
"Start_time" Register for time window 2
7
"Timestamp" Shift Register
[N] [N-1] threshold [3] [2] [1] [0]
14 17 19 20
New Event {22}
"Start_time" Register for time window 1
14
"Start_time" Register for time window 2

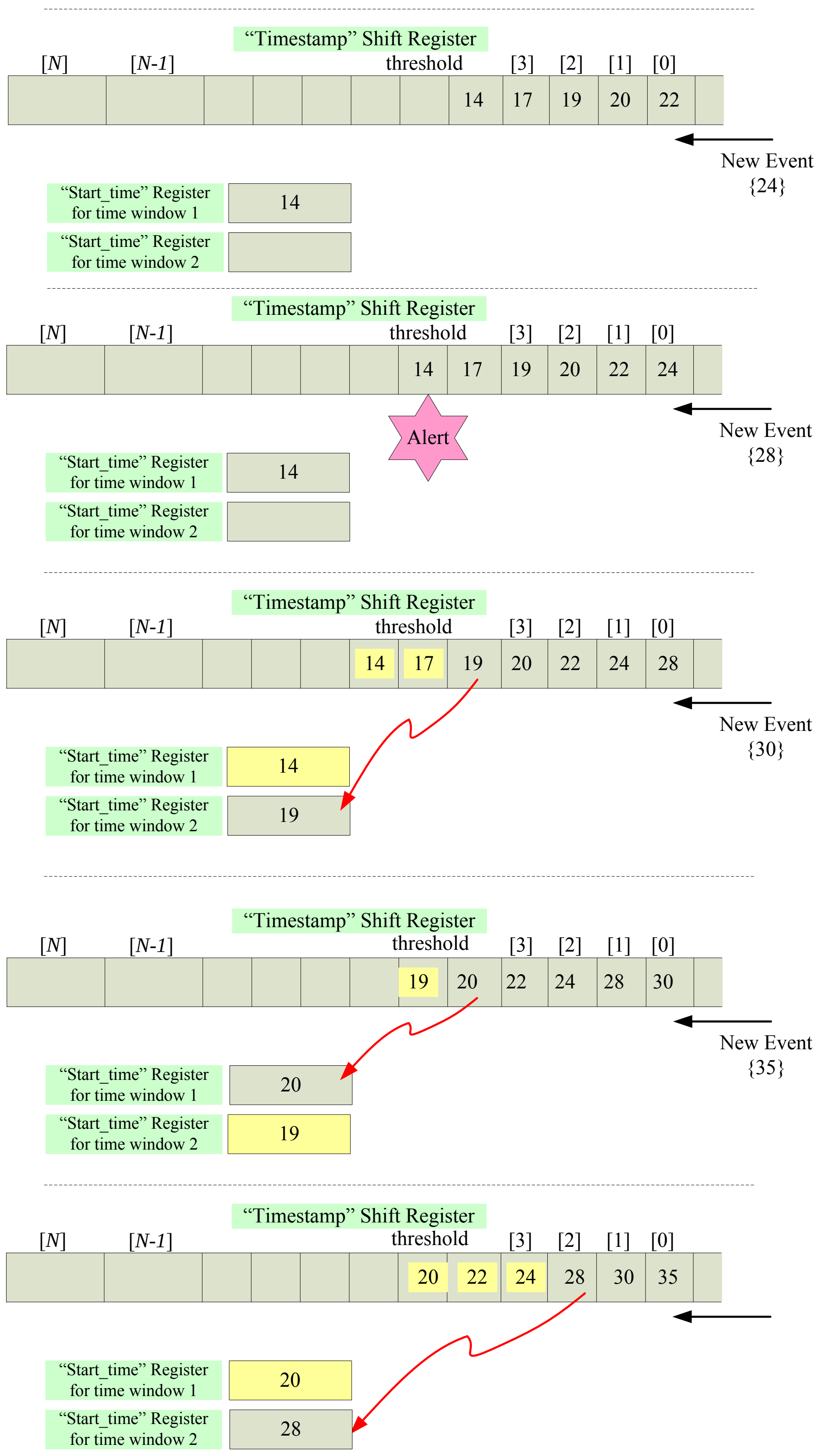

"Timestamp" Shift Register
[N]
[N-1]
threshold
[3]
[2]
[1]
[0]
14
17
19
20
22
New Event
{24}
"Start_time" Register for time window 1
14
"Start_time" Register for time window 2
"Timestamp" Shift Register
[N]
[N-1]
threshold
[3]
[2]
[1]
[0]
14
17
19
20
22
24
Alert
New Event
{28}
"Start_time" Register for time window 1
14
"Start_time" Register for time window 2
"Timestamp" Shift Register
[N]
[N-1]
threshold
[3]
[2]
[1]
[0]
14
17
19
20
22
24
28
New Event
{30}
"Start_time" Register for time window 1
14
"Start_time" Register for time window 2
19
"Timestamp" Shift Register
[N]
[N-1]
threshold
[3]
[2]
[1]
[0]
19
20
22
24
28
30
New Event
{35}
"Start_time" Register for time window 1
20
"Start_time" Register for time window 2
19
"Timestamp" Shift Register
[N]
[N-1]
threshold
[3]
[2]
[1]
[0]
20
22
24
28
30
35
"Start_time" Register for time window 1
20
"Start_time" Register for time window 2
28

Figure 5.26 Working Mechanism of Timestamp Stack in Category 2 Detection

Function 3.3: Checking Statistics with Rules

This step checks statistics value according to the rule criteria in the Rule base. Rules in the rule base are compiled original expert rules. As shown in Figure 5.27, each rule packet contains criteria, time window information, threshold criteria and rule actions with related rules.

Statistics value from the fact base has been passed into rule base execution. The number of valid incidents in the time window will be compared with the threshold field of all related rules, and find a match with one rule.

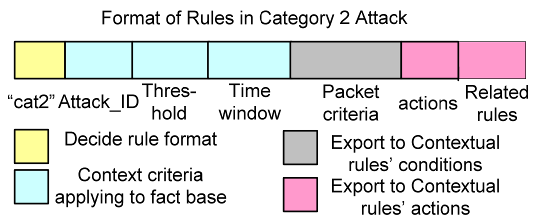

Format of Rules in Category 2 Attack
"cat2"
Attack_ID
Thres-hold
Time window
Packet criteria
actions
Related rules
Decide rule format
Context criteria applying to fact base
Export to Contextual rules' conditions
Export to Contextual rules' actions

Figure 5.27 Format of Rules in Category 2 Detection

5.6.5 Category 2 Function 4: Activating/Invalidating C-Rules

After finding a match in the rule base in Function 2, the matched rule will be used to create new Contextual Rules. A new Contextual Rules means its contextual condition has been met under the current network environment. As shown in Figure 5.28, this Contextual Rule is basically the expert system rules less the context conditions, plus some parameter information of current contexts. It also contains related rules and related fact information in its rule action field.

Section 5.4.4 described how to activate or modify this C-Rule to the C-Rule base. This function is the same for all three types of detection in multi-packet detection.

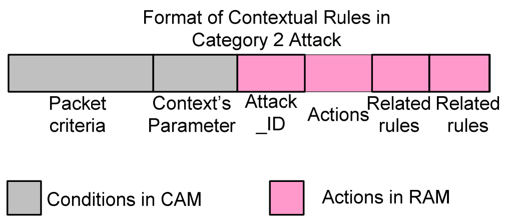


Figure 5.28 Format of C-Rules in in Category 2 Detection

5.6.6 Category 2 Function 5: Dropping Packets and Sending System Alert

System output gives output results from the expert system after inspecting the incoming packets. The output includes an action to the packet which can be one of “pass” “drop” and “log”. The output also includes an alert message to report the attack types if needed.

To implement this function, control bits for alert action will be put into RAM and an alert message number will be stored too. When a C-Rule is matched, it will read its corresponding RAM control bit to send system output including the action code to the packet, as well as an alert message.

5.6.7 Category 2 Simulation Result and Performance Analysis

For Category 2, three attacks are chosen to simulate the system. They are Smurf, Back and IPSweep attack. Simulation results are shown in Table 5.2, while the performance analysis is shown in Table 5.3.

Smurf attack is a flooding of ICMP echo reply packets destined to a victim machine. This Lincoln Lab traffic contains almost 99% malicious packets. This sniffing traffic file matches the CAM 2454 times, however, it activates only 1 new C-Rule after its first match to write the victim machine’s IP address in 2 clock cycles (20 ns) to the CAM. Matching of this new C-Rule indicates the intensity of the flooding. Besides the very low activation rate and writing CAM time, detection of Smurf requires very small CAM space as well, only two CAM entry spaces for this specific file. The low activation time allows CAIPES to keep up with wire speed detection very easily.

The same Smurf rules are applied to clean traffic which has no malicious traffic but only normal traffic including a small percentage of normal ICMP echo reply message. These ICMP triggered only two activation and four writing cycles to CAM. These two activations represent the percentage (0.0003%) of matched ICMP echo reply packet among the total number of searches in CAM. Since these ICMP packets don't have sufficient frequency to become flooding of packets, the activated C-Rules are invalidated after a period of time. The maximum number of C-Rules required here are three, 0.0003% of total number of searches in CAM during a time window.

Table 5.2 Simulation Results for Category 2 Detection

Category 2 Detection Result		Smurf		Back		IPsweep	
Input Traffic	Packets Examined	2470	20000	1952	20000	2101	20000
	Real Attack Packets Number	2454	0	1699	0	10	0
	Time Duration	11s	2123.3s	3.9s	2123.3s	90.3s	2123.3s
Search, Match	Search Number	2481	647081	4469	647081	77478	647081
	Matched Number	2454	10	1699	0	10	10
	Matched of Activated C-Rules	2453	8	1698	0	9	7
Activations	Activation Times	1	2	1	0	10	5
	Activation C-Rules	1	2	1	0	11	10
	CAM Writing Cycles	2	4	3	0	22	20
In-validate	Invalidation Times	0	2	0	0	0	5
	In-validation C-Rules	0	2	0	0	0	10
C-Rule	C-Rule Number in C-Rule Base	2	3	2	1	12	3
System Speed	Slow Updates	0	0	0	0	0	0

Table 5.3 Performance Analysis for Category 2 Detection

	Smurf and Back Attack		IPSweep Attack	
	During attack	Clean Traffic	During attack	Clean Traffic
Number of Updates of C-Rules	1	~0.0003% of number of searches in time window	Real attack packet number in time window	~0.001% of number of searches in time window
Number of C-rule Generation Per Update	1	1	1~2	2
Time of C-Rule Update Per Update	2~3 CAM cycles (20 - 30 ns)		2.2 22 ns	4 40 ns
Total Number of C-Rules in CAM	2	~0.0005% of number of searches in time window	Real attack packet number in time window	~0.0005% of number of searches in time window

Back attack is a flooding of HTTP requests with '/' so that the server doesn't know how to handle that many '/''s and will crash. It has very similar performance to the Smurf attack in CAIPES. Low activation (one activation for traffic data and no activation for clean data), few writing cycles (three clock cycles for traffic data and zero clock cycles for clean data), and few CAM entries (two C-Rules for traffic data and one C-Rule for clean data) are required when detecting Back attack in malicious traffic and clean traffic. No activation with normal traffic occurs because in this traffic file there is no HTTP request with '/'. For other normal traffic files, it could be a very few instances of this specific packet could exist which could lead to a small number of rule activation, as well as a small number of rules inside the CAM.

IPsweep is a flooding of ICMP echo request packets from a same source but to different machines in order to probe their availability. It has a Destination IP and Source IP based context, which leads to more C-Rule generations than the Smurf attack which has destination IP based context. Due to the different IP destination, each packet in the flooding will generate a new C-Rule in order to differentiate itself with other destinations. The number of newly generated C-Rules indicates the probing intensity and thus the flooding intensity. The original rule detecting any ICMP echo request packet is hit by the first packet of the flooding. This triggers two C-Rules: the first one is the same as the original rule except its source IP address is filled with the specific value of that of the first packet. This rule is put right above the original rule. The second rule is the same as the first C-Rule except its destination IP address is filled with the specific value of that of the first packet. This rule is put right above the first rule. Using the priority matching feature of the CAM, match of the first C-Rule means the ICMP echo packet comes from the same source but has a different destination. For example, when the second packet of the flooding comes, it matches the first C-Rule, and then activates a third C-Rule that has the same source IP address but different destination IP address in its fields to be put right above the 2nd C-Rule. Therefore, the number of the flooding can be counted as the match frequency of the first C-Rule in a time window. In Lincoln Lab traffic data, ten probing attempts generate eleven new C-Rules that require 22 writing cycles to CAM. The maximum CAM entries are twelve that consist of an original rule, one C-Rule with a specific source IP address, and ten C-Rules with ten different destination IP addresses, generated from probing attempts. These numbers are higher than those of the other two Category 2 attacks because IPSweep counts the number of different destination packets, instead of the number of similar packets as in Smurf and Back attack. Therefore the number of CAM entries is linear with the number of probing attempts, or the number of different ICMP echo request packets. We can limit the CAM entries number allocated to IPSweep detection with the threshold of IPsweep flooding. This prevents CAM buffer overflow without affecting the detection ability.

For IPsweep detection, CAIPES simulation with clean traffic results in similar performance as in the Smurf and Back attacks. Five out of the ten normal ping reply or request packets triggers ten C-Rules. The other five matches the C-Rules with the same source IP addresses and destination addresses and thus don't count for the flooding. These activations represent the percentage (0.001%) of matched ping packets among the total number of searches in CAM.

Because there is no real flooding involved, these C-Rules age out later and the maximum number CAM entries at any one time is low (only three). It is about 0.0005% of total number of searches in CAM during a time window.

Not counting pre-joining time in software, the CAM updating can be finished within the 200 ns packet interval time for all detection. We have a minimum of 500 ns interval time in real gigabit network. Therefore if pre-joining can be finished in 300 ns, there will be no problem for CAIPES to keep up with gigabit networks.

When a new event comes into the Pre-join block, the Pre-join block will check all related context facts and related rules to this new context and find a match between them to generate the new C-Rule. This match is O((Related_Context)*(Related_Rules)). Assume that we have 10 context facts and 30 rules related to this new context. In a Gigahertz processor, the match process would take 300 ns. In reality, a comparable number (if not fewer) of related contexts and rules are expected see in real world traffic. In addition, in Category 2 detection, a typical low number of pre-joining takes place before CAM updating will happen. Thus, CAM are updating infrequently.

In conclusion, Category 2 detection usually requires low activation, low writing cycles and small CAM space. During activation, the average C-Rule numbers to be activated at one time are 1 to 2. The average writing cycles for each C-Rule are 2 to 3. One activation requires 2-4 clock cycles. The updating time is short enough to keep up with the wire speed detection for Gigabit networks.

5.6.8 Category 2 Detection Example: Smurf Attack Detection

C-Rules for category 2 detection are mapped into CAIPES. These rules are related to the context (e.g., network status and the statistics collected from previous packets), which is the statistics fact for Category 2 detection. We use Smurf attack detection as an example to show C-Rules for the category 2 expert rules within different network contexts. Figures 5.29 to 5.5.38 shows the C-Rules detecting Smurf attack.

Figure 5.29 shows an expert system rule to detect the first flooding packet in Smurf attack. We use CLIPS [90] –like expert system rules as an example. This first line of the rule is a rule header to define a rule name and a brief explanation of the rule inside the quotation mark. The second and third lines define some variables and a pointer that w ould be used in the following rule body. A variable starts with a "$" sign, while a pointer starts with a "*" sign. An expert system has two major parts: Condition part and Action part, which are shown in the two big grey blocks in Figure 5.29, as well as included in the brackets after keywords "IF" and "THEN" respectively.

In the condition part, there are two types of condition involved in Category 2 rules, separated by the fine white lines in the gray "condition" blocks. The first type of condition is "related criteria to packet fact" which includes "packet criteria on packet fact" and "inter-factual criteria between packet fact and statistics fact". The second type of condition is "related criteria to statistics fact" which includes "statistics criteria on statistics fact" and "inter-factual criteria between statistics fact and packet fact". In Figure 5.29, conditions are expressed implicitly with the conclusion whether a specific fact currently exists or not - for example, Packet fact with protocol ICMP, code reply and destination IP equal to the statistics fact in fact base. This means "whether this packet fact exists in the network or not". The exclamation mark "!" before the statistics fact means that the condition holds true when there exists no such statistics fact.

In the action part, we use a fact pointer to store the pointer of a specific fact. "Stat1" is the pointer, and "<-" sign assign the address of a statistics fact to it. When we use "stat1" again in this rule, the specific statistics fact will be manipulated. We also define an "Output" fact where we write expert system decision to it. We perform the following action: send an alert when "alert" is '1', log this packet when "log_packet" is '1', pass this packet when "action" is '0', send this packet to a low-priority queue when "action" is '1', drop this packet when "action is

‘2’. Other than that, the expert system outputs nothing except an explicit print command to print warning message, as shown in Figure 5.33. In the action part, we also have three fields to implement the optimization of C-Rule mentioned in Section 5.3.3. The “Activate” field listed all the C-Rules that can be directly activated after a match of this C-Rule to bypass the pre-join block. The “Related_Rules” and “Related_Facts” fields listed all the facts and rules in the pre-join block that this match will relate to. This simplifies the pre-join function by limiting the facts and rules to be matched.

C-rule 1 detects the first packet of newly occurred flooding where there is no related context existing in the Pre-join block. Using the priority search ability of CAM, the function usually handled in the Pre-join block can be implemented in CAM. C-Rule 1’s criteria can be put in a CAM entry right after other context-existing C-Rule, with its Context parameter being “X (don't care)”. A match of C-Rule 1 in the CAM implies that there is no match with other C-Rules which have context information in them. This implication satisfies C-Rule 1’s context that no previous facts related to this parameter were ever recorded in the system during this period of time.

First C-rule of Smurf Detection: Establish new statistics fact when first flooding packet comes

```
Define_Rule Cat2_rule_1 “an example of Smurf attack rule --first rule”
  Variables ($var1, $time, $sys_window)
  Fact_pointer (*stat1)
  IF {
      Packet (protocol= =ICMP; ICMP_code= =reply;      //Packet criteria on Packet fact
            dst_IP= =$var1; timestamp= =$time;)    //Interfactual criteria between Packet fact and Statistics fact
       ! Statistics (Attack_ID= =smurf;   stat_value >0;             // Statistics criteria on Statistics fact
             context_para = =$var1; timestamp[0]> ($time – $sys_window); ) // Inter-factual criteria
   }
  THEN {
      Modify (Output(alert= 0; log_packet= 0; action= 0;))         //Modify alert fact to “pass” packet
      Assert (*stat1<-Statistics (attack_ID =smurf;
                                    context_para =var1; stat_numer= 1;))        //Add a stat fact
      Activate (Cat2_rule_2);                                       //Rule activation and invalidation
      Related_Rule (Cat2_rule_2};                                   //Related rules in Pre-Join block
      Related_Fact (*stat1);                                        //Related facts in Pre-Join block
  }
```

Figure 5.29 The First C-Rule in Category 2 Detection to Detect Smurf Attack

The action when C-Rule 1 is matched is to add a fact into the fact base in the Pre-join block to update the statistics. As shown in Figure 5.30, the addition of this fact will also trigger the Pre-join block to find a match with Pre-join criteria in C-Rule 2. Therefore, the system will activate a new C-rule 2 and store it in the C-Rule base. A match of C-Rule 1 involves functions of “search”, “update statistics”, “pre-joining”, and “activate new C-Rules”.

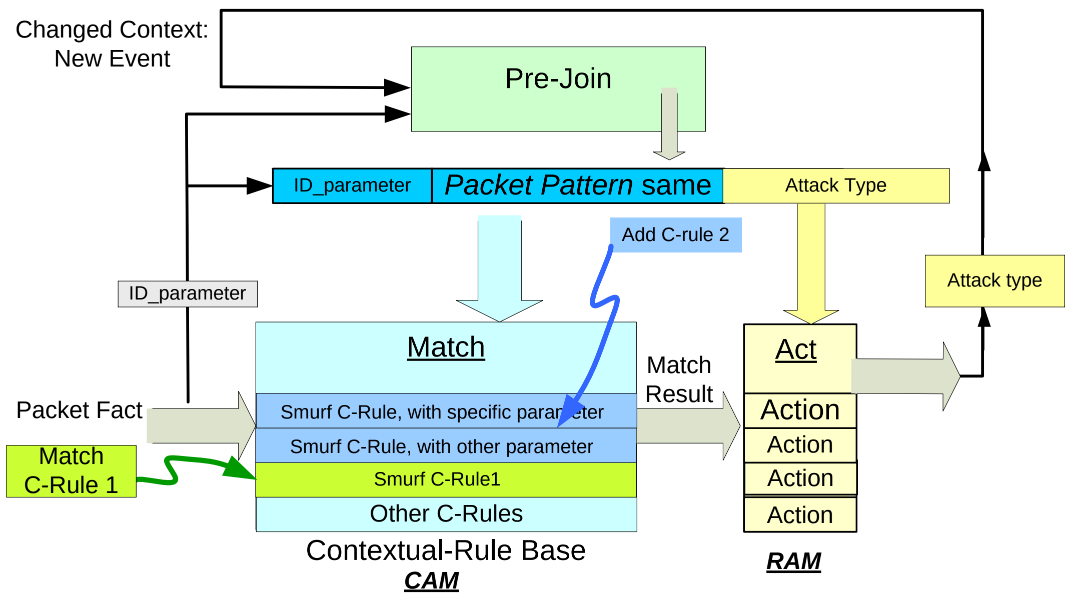


Figure 5.30 CAIPES After Match of C-Rule 1 in Category 2 Detection

Second C-rule of Smurf Detection: Counting for packets when more flooding packets come

```
Define_Rule Cat2_rule_2 "an example of Smurf attack rule --second rule"
  Variables ($var1, $time, $sys_window)
  Fact_pointer (*stat1)
 IF {
      Packet (protocol= =ICMP; ICMP_code= =reply;     //Packet criteria on Packet fact
            dst_IP= =$var1; timestamp= =$time;)     //Interfactual criteria between Packet fact and Statistics fact
      *stat1 <- Statistics (attack_ID==smurf; stat_number <N-1 && >0;          // Statistics criteria
             context_para = =$var1; start_time > ($time – $sys_window); )       // Inter-factual criteria
  }
 THEN {
      Modify (Output (alert= 0; log_packet= 0; action= 0;))          //Modify alert fact to "pass" packet
      Modify (*stat1(stat_number++;))                                //Modify the statistics fact
      Related_Rule (Cat2_rule_2, Cat2_rule_3, Cat2_rule_4)           //Related rules in Pre-Join block
      Related_Fact (*stat1)                                          //Related facts in Pre-Join block
 }
```

Figure 5.31 The Second C-Rule in Category 2 Detection to Detect Smurf Attack

When C-Rule 2 matches against the current packet, the system will add a new fact into the Pre-join fact base to update the statistics. If the Pre-join criteria in C-Rule 2 are still met (meaning the occurrence of flooding is still below threshold), the Pre-join block will do nothing, nor make any change to the C-Rule base. If the Pre-join criteria in C-Rule 2 are not met, but the C-Rule 3 Pre-join criteria are met, the system will change C-Rule 2 to C-Rule 3, where the only difference is the addition of an "alert" warning message in C-Rule 2's action part. If the C-Rule 4 Pre-join criteria are met, the system will change C-Rule 2 to C-Rule 4, where C-Rule stays the same while the fact base in Pre-join block has been updated. The functions involved in the match of C-Rule 2 include "search", "update statistics", "pre-joining" and "activate new C-Rules" when C-Rule 3 is added. As shown in Figure 5.32, no "activate new C-Rules" are needed when C-Rule 2 stays the same.

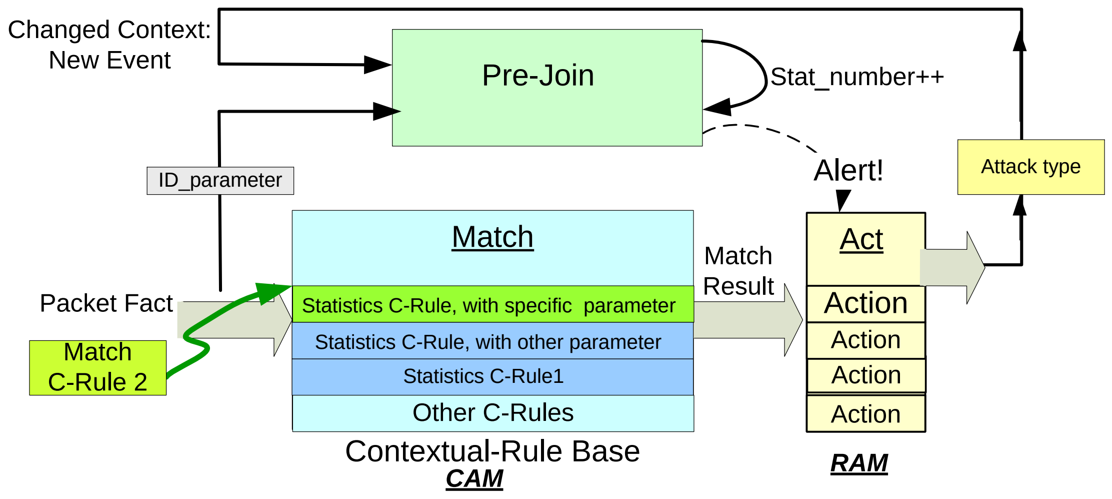


Figure 5.32 CAIPES After Match of C-Rule 2 in Category 2 Detection

Third C-rule of Smurf Detection: Alert action ready when a flooding approaching its threshold

```
Define_Rule Cat2_rule_3 “an example of Smurf attack rule --third rule”
  Variables (var1, time, sys_window)
  Fact_pointer (stat1)
  IF {
      Packet (protocol= =ICMP; ICMP_code= =reply;     //Packet criteria on Packet fact
            dst_IP= =$var1; timestamp= =$time;)     //Interfactual criteria between Packet fact and Statistics fact
     *stat1 <- Statistics (attack_ID==smurf; stat_number = = N-1;              // Statistics criteria
              context_para = =$var1; start_time > ($time – $sys_window); )      // Inter-factual criteria
   }
  THEN {
      Print (“Attack ID: Smurf, Flooding, Dropped”)                 //Print output
       Modify (Output (alert= 1; log_packet= 1; action= 2;))         //Modify alert fact to “drop” packet
       Modify (*stat1(stat_number=1;))                              // Modify the statistics fact
       Related_Rule(Cat2_rule_2)                                    //Related rules in Pre-Join block
       Related_Fact (*stat1)                                        //Related facts in Pre-Join block
  }
```

Figure 5.33 The Third C-Rule in Category 2 Detection to Detect Smurf Attack

When C-Rule 3 matches against the current packet, the system will get the alert message. The match will add a fact to the Pre-join fact base and update the statistics into one occurrence of the packets. This matches the Pre-join criteria of C-Rule 2. As shown in Figure 5.34, activation of C-Rule 2 has no changes in the CAM, but a deletion of Alert message in the RAM. The functions involved in the match of C-Rule 3 include "search", "update statistics", and "system output".

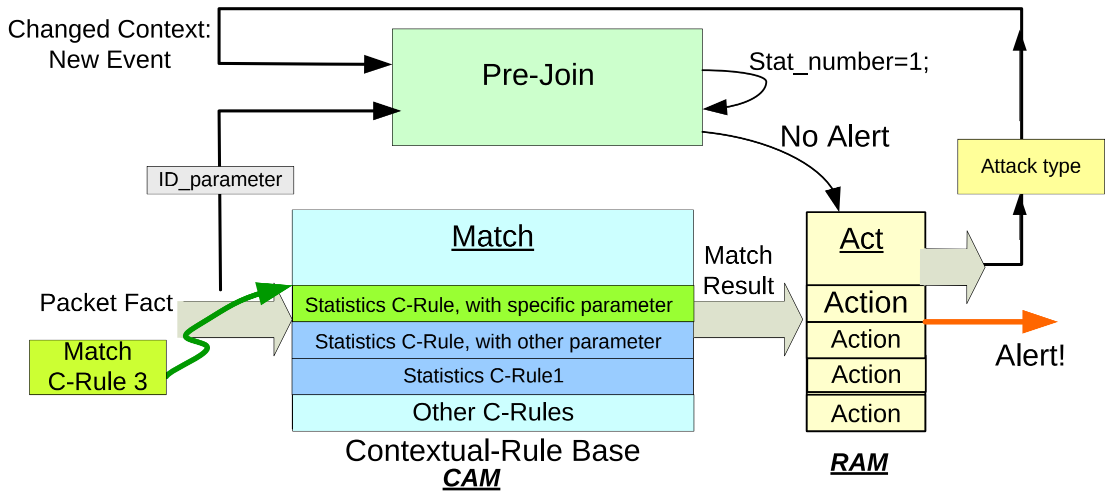


Figure 5.34 CAIPES After Match of C-Rule 3 in Category 2 Detection

Fourth C-rule of Smurf Detection: Slide new timing window when previous time window expires with packet threshold not reached

```
Define_Rule Cat2_rule_4 “an example of Smurf attack rule --fourth rule”
  Variables ($var1, $time, $sys_window, $i)
  Fact_pointer (*stat1)
  IF {
      Packet (protocol= =ICMP; ICMP_code= =reply;      //Packet criteria on Packet fact
            dst_IP= =$var1; timestamp= =$time;)     //Interfactual criteria between  Packet fact and Statistics fact
     *stat1 <- Statistics (attack_ID==smurf; stat_number<N-1 && >0;          // Statistics criteria
                        time > start_time + $sys_window && < timestamp[0] + $sys_window;
                        context_para== $var1;                                 // Inter-factual criteria
                        timestamp[$i] > $time - $sys_window;
                        timestamp[$i-1]== $new_time  && <$time - $sys_window; )
  }
 THEN {
       Modify (Output (alert= 0; log_packet= 0; action= 0;))         //Modify alert fact to “pass” packet
       Modify (*stat1(stat_number= $i+1; start_time= $new_time;
                         timestamp << $time;))                        //Modify the statistics fact
       Related_Rule(Cat2_rule_2, Cat2_rule_3, Cat2_rule_3)     //Related rules in Pre-Join block
       Related_Fact (*stat1)                                      //Related facts in Pre-Join block
  }
```

Figure 5.35 The Fourth C-Rule in Category 2 Detection to Detect Smurf Attack

When C-Rule 4 matches against the current packet, the system will add the fact to the Pre-join fact base and modify the statistics number according to the new time window. As shown in Figure 5.36, nothing else needs to be done for the system. The functions involved in the match of C-Rule 4 include “search”, “update statistics”, and “pre-joining”.

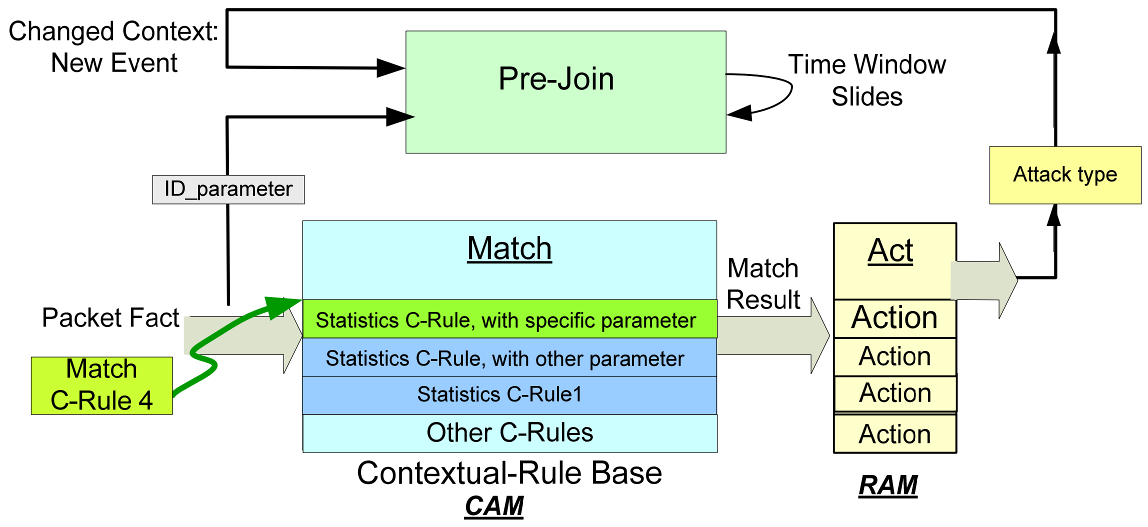


Figure 5.36 CAIPES After Match of C-Rule 4 in Category 2 Detection

Clean-up Rule in Pre-join Block: Delete the fact and the C-Rule when there is no occurrence of flooding packet for the whole time window

```
Define_Rule Cat2_rule_cleanup "an example of Smurf attack rule --cleanup rule"
  Variables ($current_time, $sys_window)
  Fact_pointer (*stat1)
  IF{
      *stat1 <- Statistics (attack_ID==smurf;   $current_time > timestamp[0] + $sys_window;)
   }
  THEN {
       Retract (*stat1)                                                  // Delete the statistics fact
       Retract (Cat2_rule_1, Cat2_rule_2, Cat2_rule_3, Cat2_rule_4)       //Delete related C-Rules
  }
```

Figure 5.37 The Cleanup C-Rule in Category 2 Detection to Detect Smurf Attack

The clean-up rule is stored in the Pre-join block. When there is no trace of a flooding packet occurring during the time window, the clean-up rule will delete the fact in the fact base and also any C-rule related to this flooding in the C-rule base, as shown in Figure 5.38. The functions involves in the match of clean-up rule includes "pre-joining" and "invalidate C-Rules".

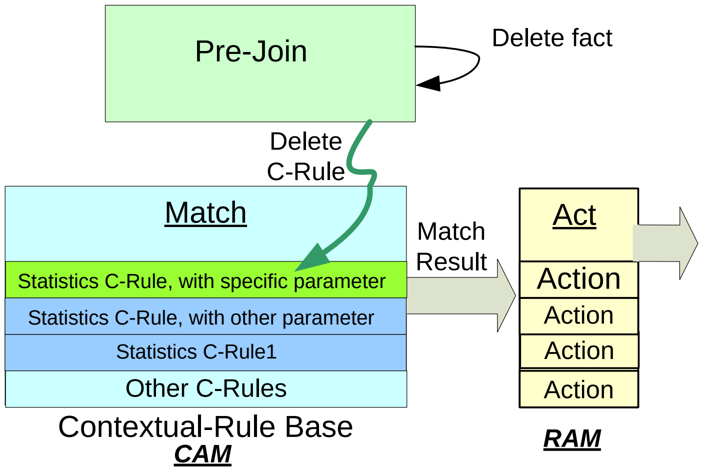


Figure 5.38 CAIPES After Match of Clean-up C-Rule in Category 2 Detection

As we can see, C-Rules 2, 3 and 4 have the same packet criteria in the C-Rule base. There are no dynamic changes in the C-rule base criteria part when switching from C-Rule 2, 3, and 4. The only change occurring in the C-Rule action part is from Rule 2 to Rule 3. It has only one occurrence among the flooding threshold number of packets (whish typically is tens or hundreds) and happens on the action part RAM, instead of CAM.

Matches of C-Rule 1 and clean up C-rule are the only rules bring addition or deletion to the CAM respectively. Again, it has only very little occurrence among the flooding threshold number of packets (whish typically is tens or hundreds). C-Rules 1, 2, 3, 4 and Clean-up Rule are generated under different network contexts. The five different network contexts cover all possibilities in reality, as shown in Figure 5.39.

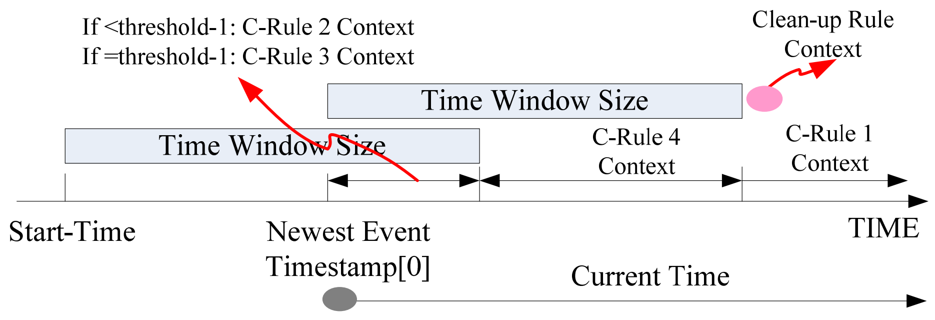


Figure 5.39 Time Coverage from Different C-Rules in Category 2 Detection

Therefore, Category 2 C-rules in a priority CAM have the following features:

1. C-Rules 1 searching for new facts (e.g. C-Rule 1) can be put in a CAM entry right after those C-Rules that are Pre-joined with existing facts (e.g. C-Rule 2, 3 and 4);
2. When the first packet of the flooding comes, a new C-rule (C-Rule 2) that are Pre-joined with a specific parameter will be added into C-Rule base.
3. Then C-Rule 2, 3, 4's criteria in CAM remains the same for that instance of attacks.
4. When the flooding disappears for some time, the specific C-Rule will be deleted from the C-Rule base.

5.6.9 Category 2 Detection Conclusion

Category 2 attack can be implemented in the C-Rule method with CAM and RAM. It requires very few numbers of CAM modifications and each modification requires very few writing cycles. Detection towards three attacks from Lincoln Lab data has been implemented in a prototype system and it can keep up with gigabit network traffic.

5.7 DETECTION OF CATEGORY 3 NETWORK INTRUSIONS

Category 3 attack is multi-packet attack that follows a specific packet sequence. This sequence of packet is malicious and do harm to networks and computers.

Problem: There are multi-packet attacks that follow a specific packet sequence and eventually cause damage to networks or computers, e.g., machine rebooting or information exposition. An analysis of the Lincoln Lab IDS evaluation dataset shows that 39% of multi-packet attacks in are Category 3 attacks. These attacks are enumerated in Table 5.1.

Example: An example of Category 3 attack is the Queso probing. Queso is a probing utility used to determine the type of machine and operating system for a given IP address. Queso sends a sequence of 7 TCP packets each with different flags set to one port of a machine, as shown in Figure 5.40. Since different machines' implementations and operation systems will handle these packets differently, returned packets will be used to decide operation system types.

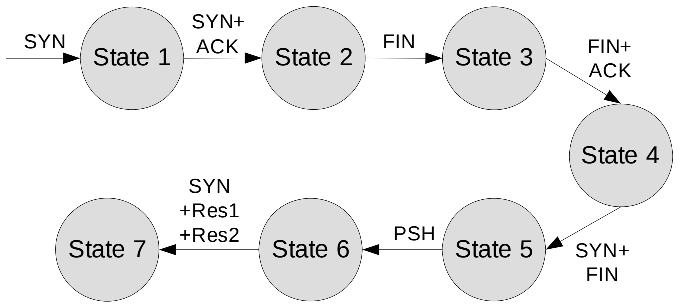


Figure 5.40 Attacking Mechanism of in Category 3 Queso Attack

Detection: To detect category 3 attacks, the expert system analyzes multiple packets and their arrival sequences. It detects malicious traffic in Category 3 by recognizing a featured pattern not inside a single packet, but inside the sequence of packets. It is able to detect multiple-packet attacks, where each packet in a group seems normal but they constitute an attack when taken together with the other members of the group. Furthermore, it can detect anomalies that violate a defined sequence pattern in a protocol.

Supported detection includes:

- **Multi-packet signature matching:** The state transition model describes an attack with its multiple packets and their sequence;

- **State transition validation in a protocol (e.g., TCP):** The state transition model describes a normal transition sequence in a protocol. This detects violations of a normal transition sequence defined in a protocol.

5.7.1 Expert System Functions Required for Category 3 Detection

To recognize a Category 3 attack, expert system rules describing it check the state transition from specific packets during a period of time. The context fact in Category 3 detection is *"State fact"* where the state number, state connection parameters and timestamp of the event occurrence are included. The rule condition in Category 3 has *"State Criteria"* as its context criteria to check on the state facts.

The functions required for Category 3 attacks are listed in the order of how a final attack state is detected as follows:

1. **Packet Searching C-Rule Base:** Packet Searches C-Rules, finding the first packet of the state transition
2. **Modify State Facts (Optional):** Modify state facts in pre-join block
3. **Pre-joining in Pre-join Block for States Checking (Optional):** Check if the state criteria of any rules are satisfied
4. **Activate New C-Rule:** Activating a new C-Rule that targeting to a new state. The C-rule embeds with the state's context information.
5. **Repeat Step 1-4** if the attack doesn't reach the final state
6. **Dropping Packets and Sending System Alert** when the attack reaches the final attack state
7. **Invalidate C-Rules:** Invalidate any C-Rules related to this connection

Step 2 and 3 can be skipped when a direct activation from a C-Rule is valid. Usually in a state transition driven only by a network packet, a direct activation from a C-Rule can be executed. This is because when this C-Rule is matched, the activated C-Rule's context criterion is satisfied. When the state transition requires not only a network packet, but also other network context, a match of the first C-Rule couldn't directly activate new C-Rules. Pre-joining is required to check if other network context criteria are satisfied or not.

Therefore, five functions are required in Category 2 attacks. They are "Searching C-Rule base", "Modifying State Facts", "Pre-joining", "Activating C-Rules", and "Dropping Packets and Sending System Alert". Details of these functions are described in the following sections:

5.7.2 Category 3 Function 1: Searching C-Rule base

This function is the basic function of the C-Rule method. It is the same for all three types of multi-packet attacks. This function has been described in Section 5.4.1.

5.7.3 Category 3 Function 2: Modifying State Facts

Changed network context is generated from a matched C-rule. Each coming changed network context can be seen as an event. In category 3 attacks, an event is a newly generated state. It includes information of the attack ID, the parameters to identify the incident, the state number, the timestamp, the related rule number and related fact number.

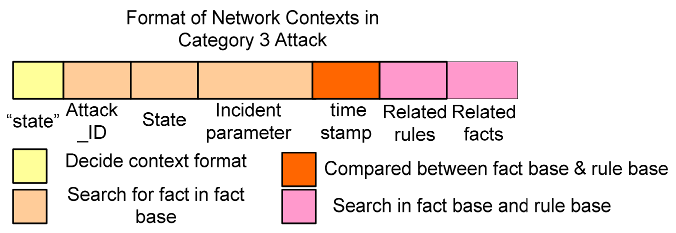

Format of Network Contexts in Category 3 Attack
"state"
Attack _ID
State
Incident parameter
time stamp
Related rules
Related facts
Decide context format
Search for fact in fact base
Compared between fact base & rule base
Search in fact base and rule base

Figure 5.41 Format of State Fact in Category 3 Detection

5.7.4 Category 3 Function 3: Pre-joining for States Check

Category 3 rules describe the transition condition for state transitions. Usually most of the Category 3 rules can directly activate its following rules of the next possible transitions to overwrite themselves. Only complicated Category 3 rules requires the pre-join function to check the timing issue and combine other state transitions to match a rule condition. Figure 5.42 shows the details of this pre-joining. Changed state facts go into the Pre-join block and are stored in the fact base according to the "related fact" field. A time window criterion in the fact base will be exercised on this new fact to make sure this state transition happens within the valid time window. If it is satisfied, the new fact will be compared with related rules in rule base and find a match. The rule base writes back the new time window information to the fact base and also generates new C-Rules.

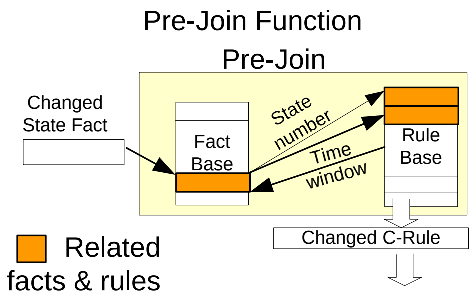


Figure 5.42 Pre-join Block in Category 3 Detection

Function 3.1: Store New State in Pre-join block

The Pre-join fact base saves those changed state facts. All related time window information for transitions from that state is extracted from rules and is saved in the stack, for simpler execution of Pre-joining between the rule base and the fact base. The related C-Rule address is also stored for later fast deletion of the C-Rule in C-Rule base.

When a new network state fact comes in, the Pre-join function will first read its “related fact number” field, and find the state with the same attack ID and parameters. If such a state exists, the timestamp of the new event will be compared to see if it is within its time window to enable the transition. Once confirmed it is a valid transition, the state and time stamp field in the fact will be overwritten by the new state number and new time stamp. The new state number information is passed to rule base execution.

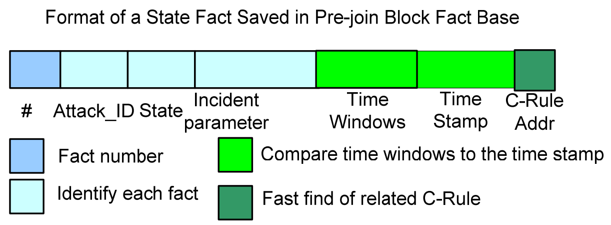
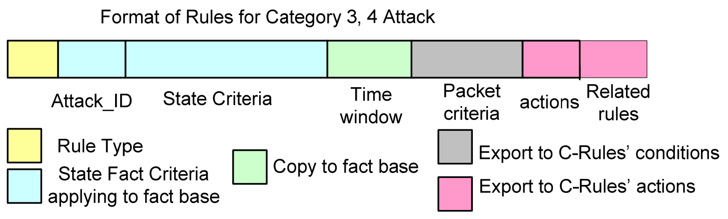

Format of a State Fact Saved in Pre-join Block Fact Base
#
Attack_ID
State
Incident parameter
Time Windows
Time Stamp
C-Rule Addr
Fact number
Identify each fact
Compare time windows to the time stamp
Fast find of related C-Rule

Figure 5.43 Format of State Facts Stored in Pre-join Block in Category 3 Detection

Function 3.2: Check State in Pre-join block

Rules in rule base are compiled original expert rules. Each rule includes packet criteria, context criteria, inter-factual criteria and rule actions. For Category 3 and 4 attacks, it includes packet criteria, state criteria, time window information, and rule actions with related rules.

When fact base execution has finished, information from the fact base has been passed into rule base execution. The new state number will be compared to all related rules. After that, the fact base will be searched for other state criteria in matched rules if necessary. If these state criteria are also matched, this rule is matched with current network context. The time window in this rule will overwrite the "time window" field in the matched fact in the fact base.



Format of Rules for Category 3, 4 Attack
Attack_ID
State Criteria
Time window
Packet criteria
actions
Related rules
Rule Type
State Fact Criteria applying to fact base
Copy to fact base
Export to C-Rules' conditions
Export to C-Rules' actions

Figure 5.44 Format of Rules in Category 3 Detection

5.7.5 Category 3 Function 4: Activating /Invalidating new C-Rules

After a match is found in the rule base, the matched rule will be used to create new Contextual Rules. A new Contextual Rule means its contextual condition has been met under the current network environment. This Contextual Rule is basically the expert system rules less the context conditions, plus some parameter information of current contexts. It also contains related rules and related fact information in its rule action field.

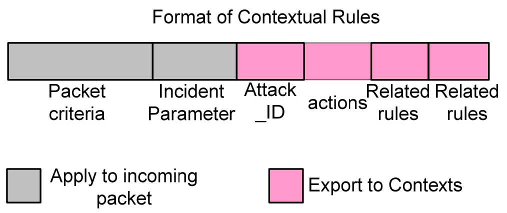


Figure 5.45 Format of C-Rules in Category 3 Detection

In Category 3 detection, when one C-Rule is matched, it usually activates another C-Rule or C-Rules to replace it. This ocurs because these C-Rules represent the transition conditions from state to state, and one transition state needs to be replaced by the transition state of the next state. Therefore, activating new C-Rules usually overwrites the old C-Rules and replaces them. Invalidating C-Rules happens when a state transition has reached its final state and therefore no more transition is expected for this context fact. The context related C-Rules are then invalidated and free up the CAM for future use.

Section 5.4.4 described how to activate or modify this C-Rule to the C-Rule base. This function is the same for all three types of detection in multi-packet detection.

5.7.6 Category 3 Function 5: Dropping Packets and Sending System Alert

This is the same as Category 2 detection when a real attack happens. The details have been described in Section 5.6.6.

5.7.7 Category 3 Simulation Result and Performance Analysis

For Category 3, three attacks are chosen to simulate the system. They are Queso, Teardrop and Netcat attack. Simulation results are shown in Table 5.4, while the performance analysis is shown in Table 5.5.

Queso detection detects state transitions from the 7 TCP packets, each set with different flag. Since the first of the seven packets are the same with the first TCP connection packets, normal TCP connection will trigger Queso C-Rules too, but they never transit to the final state. Therefore, we simulate the traffic with two detections: Queso rule alone detection, as well as Queso and TCP rules together detection. Since these two transitions are all connection-based, the updating number and C-Rule number are not low. The Queso rule alone case has 17 activations (0.20% of searches) and one C-Rule per activation. An average of 2.7 writing cycles of each C-Rule happens here. The Queso and TCP rule together case has a busier C-Rule base with more updates (29, 0.33% of searches), more C-Rules per activation (1.4), and more writing cycles to the CAM per C-Rules (2.9). Since most state transitions reach their final state, the C-Rules related to these transitions are eventually invalidated. Therefore only a small CAM size is required for C-Rules at any one time. In our simulation, the maximum C-Rule base requirement is 3 in the former case and 5 in the latter case.

Table 5.4 Simulation Results for Category 3 Detection

Category 3 Detection		Queso			Teardrop		NetCat		
Input Traffic	**Packets Examined**	1890		2001	102	2001	4575		2001
	Real Attack Packets Number	7		0	90	0	64		0
	Time Duration	50.5s		207.7s	1.1s	207.7s	89s		207.7s
		Rules Alone	With TCP Rules	With TCP Rules			Rules Alone	With TCP Rules	With TCP Rules
Search, Match	**Search Number**	8593		56684	363	56684	115547		56684
	Matched Number	18	41	372	315	0	6	747	372
	Matched of Activated C-Rules	6	29	248	135	0	5	499	248
Activations	**Activation Times**	17	29	248	45	0	5	499	248
	Activation C-Rules	17	41	372	45	0	5	499	248
	CAM Writing Cycles	46	118	1196	225	0	31	1513	744
In-validate	**Invalidation Times**	1	17	124	45	0	1	248	124
	In-validation C-Rules	1	33	161	45	0	1	248	124
C-Rule	**C-Rule Number in C-Rule Base**	3	5	40	2	1	2	4	2
System Speed	**Slow Updates**	0	0	0	0	0	0	0	0

Table 5.5 Performance Analysis for Category 3 Detection

	Queso and Netcat Attack		**TearDrop Attack**	
	During attack	**Clean Traffic**	**During attack**	**Clean Traffic**
Number of Updates of C-Rules	~ 0.33-0.43% of number of searches in time window (with both attack rules and TCP rules)		Real attack packet number in time window	
Number of C-rule Generation Per Update	1-1.5		1	
Time of C-Rule Update Per Update	3~6 CAM cycles (30 - 60 ns)		5 CAM cycles 50 ns	
Total Number of C-Rules in CAM	intermediate state in time window <100		intermediate sate in time window <10	

Queso detection in clean traffic leads to many more open state transitions that start as a normal TCP connection but never reach the Queso final state. This brings similar updating performance but more C-Rules in the CAM at the same time. Therefore, clean traffic leads to similar number of C-Rule base activations (248, 0.43% of searches), similar C-Rules per activation (1.5), and similar writing cycles to the CAM per C-Rules (3.2). However, the CAM size required for C-Rules at one time is 40 C-Rules, much more C-Rule base space in normal traffic than when under attack.

Teardrop attack consists of two consecutive UDP packets with misfragmentation. In order to detect the misfragmentation, we apply the novel encoding scheme for range matching invented in Chapter 4 into the header fields for fragment offset range criteria. Because of its few states and simple transition, the C-Rule per activation is low (1), and the CAM space required is low as well (2, one for original rule, one for C-Rule). The average writing cycle per C-Rule is as high as 5, because of its range encoding field being very wide to be written in every activation.

In clean traffic, there is no such mis-fragmented packet existing. Therefore, all numbers related to the C-Rule updating is 0. The C-Rule base requires only 1 C-Rule in it.

Netcat attack consists of several packets doing telnet session on a domain (53) port. The specific Netcat rule will be activated by those specific packets which are a small number of traffic. Therefore the activation times (0.0043% of searches) and C-Rule per activation (1) is very low. However, the writing cycles per C-Rule is high (average >6), since the C-Rule contains payload string matching and thus needs more time to be written.

If we combine TCP detection rules, the system will be updating a lot more due to the TCP connections in the traffic, and the activation number (0.43% of searches) goes up steeply. This is because TCP connection is very common in the traffic and it has a connection based context. The C-Rule number per activation remains the same (1). The writing cycles per C-Rule goes down to an average of 3 since now TCP connection transition causes most C-Rule updating.

In clean traffic, no Netcat attack involved thus no Netcat transition happens. Normal TCP connection leads to the C-Rule updating. Therefore, it has very similar performance as the previous column: 0.44% of searches resulting in C-Rule activation, and one C-Rule per activation, and 3 writing cycles per C-Rule.

However, for Netcat detection, since most of the transitions in the traffic reach their final state, the CAM entries required to keep up with any ongoing connection are few (2-4).

All these detection rules directly activate the next possible transition rules without pre-joining. Therefore, the C-Rule updating time is just the CAM updating without software pre-joining execution time. Experiments in Category 3 detection shows it can keep up with Gigabit detection.

In conclusion, Category 3 detection usually requires moderate to high activation, long writing cycles and variable CAM space. If the traffic contains many open transitions which will never reaches the detection rule's final state, the CAM space required is high. Otherwise, CAM space requirement is low (<10) and can easily be satisfied. The average C-Rule numbers per C-Rule activation are 1 to 1.5, and the average writing cycles for every C-Rule is 3 to 6. One activation requires 3-6 clock cycles. The updating time is short enough to keep up with wire speed detection for Gigabit networks.

5.7.8 Category 3 Detection Example: Queso Attack Detection

C-Rules for category 3 will be mapped into CAIPES. These rules are related to the context (e.g., network status and the statistics collected from previous packets). We use Queso attack detection as an example to show C-Rules for the category 3 expert rules within different network contexts. C-Rules for the category 3 expert rules for different network contexts are shown in Figures 5.46 to 5.53.

In each rule, there are two parts: the condition part and the action part which are shown in the two big gray blocks. In the condition part of Category 3 rules, there are two types of conditions involved, separated by the fine white lines in the gray "condition" blocks. The first type of condition is "related criteria to packet fact". This includes "packet criteria on packet fact" and "inter-factual criteria between packet fact and state fact". The second type of condition is "related criteria to state fact". This includes "state criteria on state fact" and "inter-factual criteria between state fact and packet fact".

In the action part, we use a fact pointer to store the pointer of a specific fact. "State1" is the pointer, and "<-" sign assigns the address of a state fact to it. When we use "state1" again in this rule, the specific state fact will be manipulated.

First C-rule of Queso Detection: Establish a new state 1 for Queso attack

```
Define_Rule Cat3_rule_1 "an example of Queso attack rule --first rule"
  Variables ($src, $dst, $src_port, $dst_port,  $time, )
  Fact_pointer (*state1)
  IF {
     Packet (protocol== TCP; TCP_flag= = SYN;            //Packet criteria on Packet fact
         src_IP==$src; dst_IP==$dst;                     //Interfactual criteria between Packet fact and State fact
         src_port = = $src_port; dst_port = =$dst_port;  timestamp= =$time;)
     ! State (attack_ID= =queso;                         //State criteria on state fact
          context_para= =$src $dst $src_port $dst_port; )   //Inter-factual criteria
   }
 THEN {
     Modify (Output (alert= 0; log_packet= 0; action= 0;))    //Modify alert fact to "pass" packet
     Assert (*state1<-State(attack_ID=queso; state_number=1;  //Add new state fact
               context_para= $src $dst $src_port $dst_port;   timestamp = $time;)
     Activate Cat3_rule_2;                               //Rule activation and invalidation
     Related_Rule (Cat3_rule_2)                          //Related rules in Pre-Join block
     Related_Fact (*state1)                              //Related facts in Pre-Join block
  }
```

Figure 5.46 The First C-Rule in Category 3 Detection to Detect Queso Attack

C-rule 1 detects the first packet of newly established state transitions where there is no related context existing in the Pre-join block. Using the priority search ability of CAM, the function usually handled in the Pre-join block can be implemented in CAM. C-Rule 1's criteria can be put in a CAM entry right after other context-existing C-Rule, with its Context parameter being "X (don't care)". A match of C-Rule 1 in the CAM implies that there is no match with other C-Rules which have context information in them. This implication satisfies C-Rule 1's context that no previous facts related to this parameter were ever recorded in the system during this period of time.

The action when C-Rule 1 is matched is to add a fact into the fact base in the Pre-join block to update the statistics. As shown in Figure 5.47, the addition of this fact will also trigger the Pre-join block to find a match with Pre-join criteria in C-Rule 2. In fact, the system can activate a new C-rule 2 directly and write it in the C-Rule base. A match of C-Rule 1 involves functions of "search", and "activate new C-Rules".

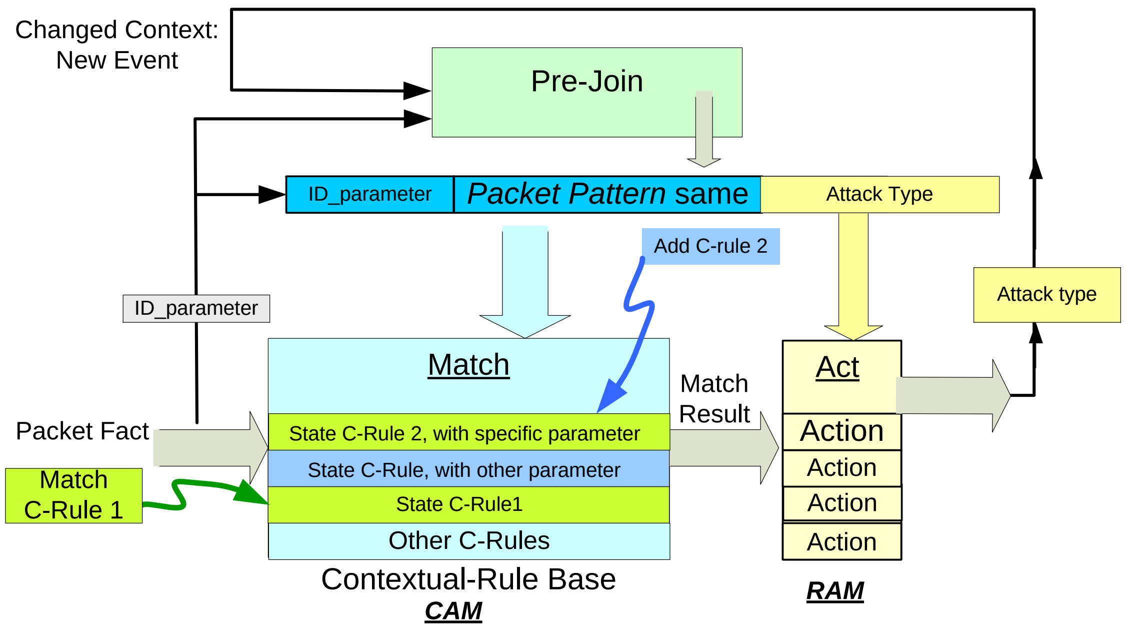


Figure 5.47 CAIPES After Match of C-Rule 1 in Category 3 Detection

Second C-rule of Queso Detection: transition from state 1 to state 2

```
Define_Rule Cat3_rule_2 “an example of Queso attack rule --second rule”
  Variables ($src, $dst, $src_port, $dst_port,  $time, )
  Fact_pointer (*state1)
  IF {
      Packet (protocol== TCP; TCP_flag= = SYN+ACK;              //Packet criteria on Packet fact
          src_IP==$src; dst_IP==$dst;                  //Interfactual criteria between Packet fact and State fact
          src_port = = $src_port; dst_port = =$dst_port;  timestamp= =$time;)
      *state1 <- State (attack_ID= =queso;    state_number==1;           //State criteria on state fact
                  context_para= =$src $dst $src_port $dst_port; )        //Inter-factual criteria
   }
  THEN {
      Modify (Output (alert= 0; log_packet= 1; action= 0;))      //Modify action to “send to queue”
      Modify (*state1 (state_number= 2; timestamp = ?time;) )    //Modify the state fact
      Activate Cat3_rule_3;                                      //Rule activation and invalidation
      Related_Rule (Cat3_rule_3)                                 //Related rules in Pre-Join block
      Related_Fact (*state1)                                     //Related facts in Pre-Join block
  }
```

Figure 5.48 The Second C-Rule in Category 3 Detection to Detect Queso Attack

When C-Rule 2 matches against the current packet, the system will add a new fact into the Pre-join fact base. As shown in Figure 5.49, this new fact will trigger the Pre-join block to activate a new C-Rule 3. In fact, the system can activate a new C-rule 3 directly and write it in the C-Rule base. This C-Rule 3 will overwrite the C-Rule 2 in C-Rule base. A match of C-Rule 2 involves functions of "search", and "activate new C-Rules".

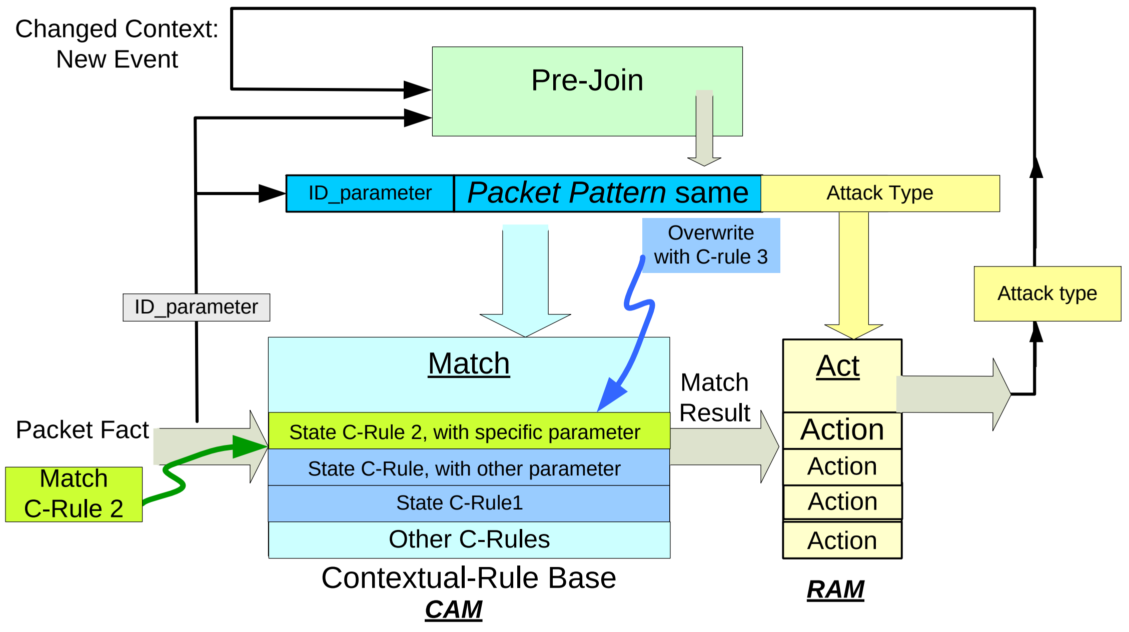


Figure 5.49 CAIPES After Match of C-Rule 2 in Category 3 Detection

We replicated the same C-Rule modification for C-Rules 3, 4, 5, 6 which are C-Rules to transmit from State 3 to State 4, State 4 to State 5, and State 5 to State 6. Each C-Rule only differs from the condition where the TCP_flag are with various bits set.

When C-Rule 6 matches against the current packet, the system will add a new fact into the Pre-join fact base. As shown in Figure 5.50, this new fact will trigger the Pre-join block to activate a new C-Rule 7. This C-Rule 7 is targeted for final attack state 7 so it has an "alert" warning message and a "drop" action in its RAM control bits. In fact, the system can activate a new C-rule 6 directly and write it in the C-Rule base. C-Rule 7 will overwrite C-Rule 6 in the C-Rule base. A match of C-Rule 6 involves functions of "search", and "activate new C-Rules".

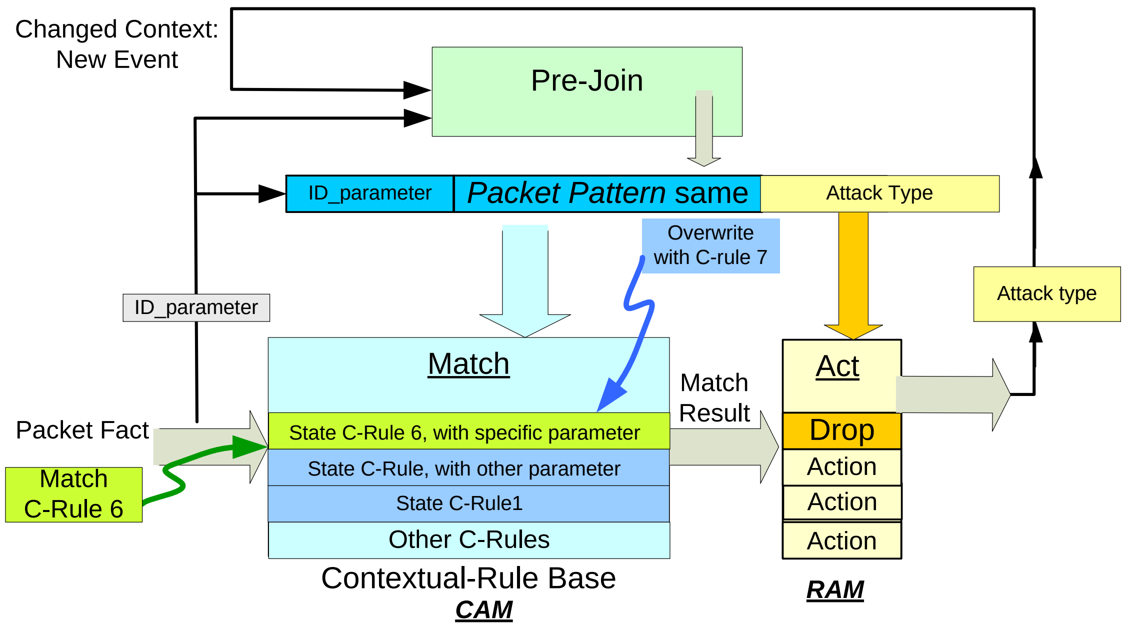


Figure 5.50 CAIPES After Match of C-Rule 6 in Category 3 Detection

Seventh C-rule of Queso Detection: transition from state 6 to state 7

```
Define_Rule Cat3_rule_2 “an example of Queso attack rule --second rule”
  Variables ($src, $dst, $src_port, $dst_port,  $time, )
  Fact_pointer (*state1)
  IF {
     Packet (protocol== TCP; TCP_flag= = SYN+Res1+Res2;            //Packet criteria on Packet fact
         src_IP==$src; dst_IP==$dst;                //Interfactual criteria between Packet fact and State fact
         src_port = = $src_port; dst_port = =$dst_port; )
     *state1 <- State (attack_ID= =queso;    state_number==6;       //State criteria on state fact
                 context_para= =$src $dst $src_port $dst_port; )     //Inter-factual criteria
   }
 THEN {
     Print (“Attack ID: Queso, Probing,  Dropped”)
     Modify (Output (alert= 1; log_packet= 1; action= 2;))        //Modify alert fact to “drop” packet
     Retract (*state1)                                            //Delete the state fact
     Invalidate Cat3_rule_7;                                      //Rule activation and invalidation
 }
```

Figure 5.51 The Seventh C-Rule in Category 3 Detection to Detect Queso Attack

A match of C-Rule 7 means the attack reaches the final state. As shown in Figure 5.52, the Pre-join block will then invalidate this C-Rule because it is already the final state and has no other state to reach. A match of C-Rule 7 involves functions of "search", and "Invalidate new C-Rules" and "system output".

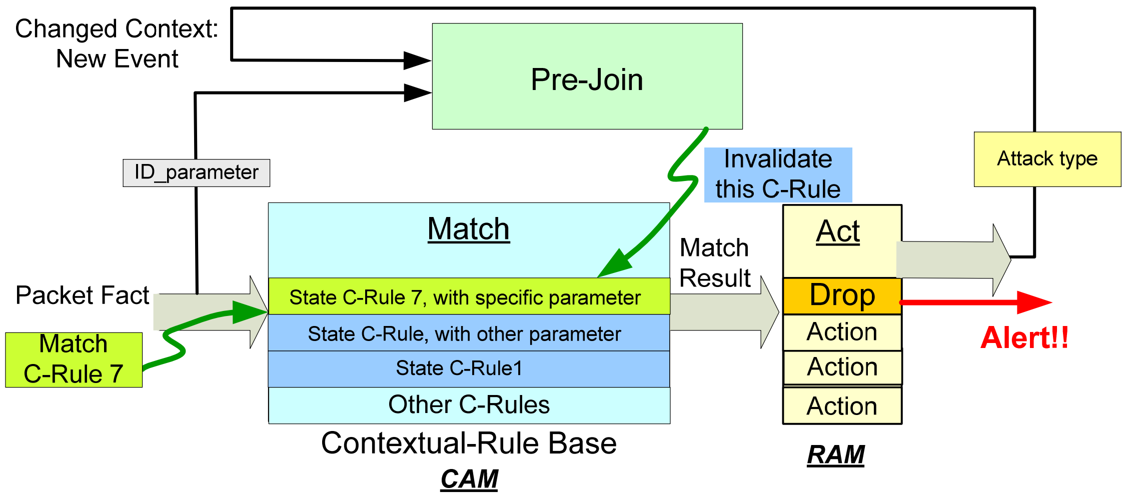


Figure 5.52 CAIPES After Match of C-Rule 7 in Category 3 Detection

Clean-up C-Rule of Queso Detection: Delete the related facts and rules when there is no state transition for a long time

```
Define_Rule Cat3_rule_cleanup "an example of Queso attack rule --cleanup rule"
  Variables ($src, $dst, $src_port, $dst_port, $current_time, $threshold )
  Fact_pointer (*state1)
 IF {
     *state1 <- State (attack_ID= =queso;                          //State criteria on state fact
                   $current_time –  timestamp > $threshold;
  }
 THEN {
     Retract (*state1)                                        //Modify the state fact
     Invalidate ( Cat3_rule_1, Cat3_rule_2, Cat3_rule_3, Cat3_rule_4,
                    Cat3_rule_5, Cat3_rule_6, Cat3_rule_7;)        //Rule activation and invalidation
 }
```

Figure 5.53 The Cleanup C-Rule in Category 3 Detection to Detect Queso Attack

The clean-up rule is stored in the Pre-join block. When there is no state transition occurring for a long time, the C-Rule will age. As shown in Figure 5.54, the clean-up rule will delete the fact in the fact base and also invalidate the C-rule in the C-rule base. The functions involves in the match of clean-up rule includes "invalidate C-Rules".

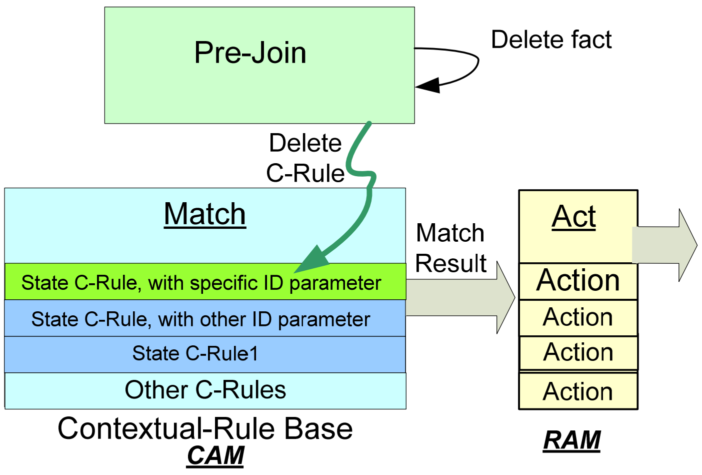


Figure 5.54 CAIPES After Match of Clean-up C-Rule in Category 3 Detection

5.7.9 Two or More Possible Transitions From One State in Category 3

Consider there is also a TCP connection transition rules in the expert system which is shown in Figure 5.55. Therefore, after a SYN packet is seen in the network, there are two possible transitions: either in queso state transition or TCP connection state transition. The first transition will be fulfilled when a SYN+ACK packet comes from the same computer as the first SYN packet. The second transition will be fulfilled when a SYN+ACK packet comes from the destination computer in the first SYN packet.

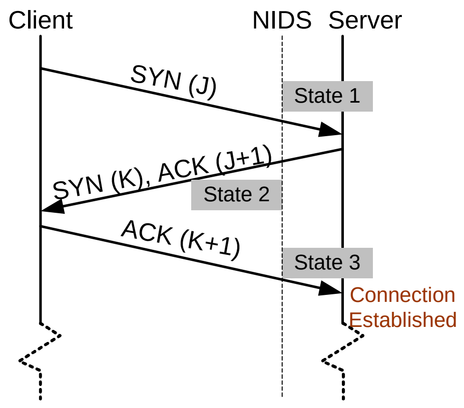


Figure 5.55 Establishing a TCP Connection

Therefore when C-Rule 1 is matched, Pre-join will actually add two C-rules in the C-Rule Base as shown in Figure 5.46: C-rule 2 in the queso state transition diagram, and C-Rule 2 in the TCP connection state diagram.

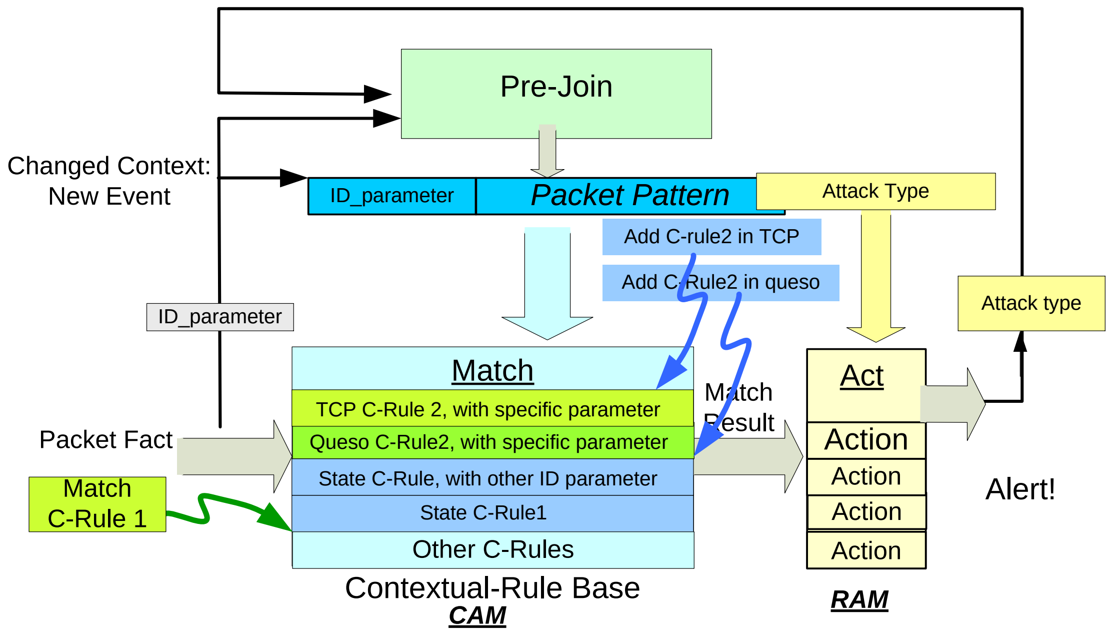


Figure 5.56 Adding Two Possible Transitions in C-Rule Base Expert System in Category 3 Detection

The features of the C-Rules of this State Transition Tracking architecture are as follows:

- Every new match of a C-Rule in Rule Base will cause the Pre-join to generate new C-Rules leading to the next state.
- Old C-Rules which will never generate any matches will be deleted or overwritten in time.

5.7.10 Category 3 Detection Conclusion

Category 3 attack can be implemented in the C-Rule method with CAM and RAM. It requires moderately high numbers of CAM modifications and each modification requires 3-6 writing cycles. Detection towards three attacks from Lincoln Lab data has been implemented in a prototype system and it can keep up with gigabit networks.

5.8 DETECTION OF CATEGORY 4 NETWORK INTRUSIONS

Category 4 attack is multi-packet attack containing flooding of a certain packet sequence. This flooding of sequence is malicious and do harm to networks and computers.

Problem: There exist multi-packet attacks in which the sequence of packets is harmless, but the high frequency of occurrence of this sequence leads to damage to the network, e.g., Denial of Service. An analysis of the Lincoln Lab IDS evaluation dataset shows that 39% of multi-packet attacks in are Category 4 attacks. These attacks are enumerated in Table 5.1.

Example: An example of Category 4 attack is the SYN flooding attack. In a SYN flooding attack, an attacker sends many SYN packets to the victim but intentionally withdraws the final acknowledge ACK packets that are required to establish a TCP connection (shown in the dark gray circles in Figure 5.57). This leads to a huge number of half-open pending connections and will finally exhaust the victim's "pending connection" recording memory. After this happens, the victim can no longer set up TCP connections as there will not be space in the pending connection memory until these pending connections time out.

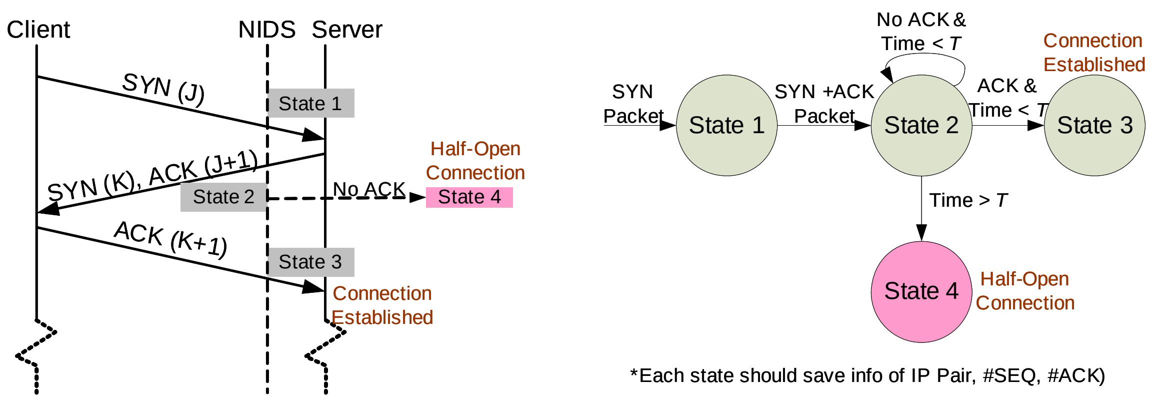


Figure 5.57 Handshake Communications in TCP Connection

Detection: This architecture analyzes multiple sequences of packets and their occurrence frequency statistics. It detects malicious traffic in Category 4 that has a featured pattern not inside a single packet or a sequence of packets, but in its sequence occurrence frequency. It is able to detect flooding multi-packet attacks where each multi-packet sequence seems normal but their frequency indicates that they are part of an attack. Finally, it can report the severity of an intentional attack by reporting the number of occurrences of a specific sequence.

Detection methods that this architecture supports include:

- **Threshold anomaly detection:** Setting a threshold value for the occurrence frequency of a specific sequence. Traffic going beyond the threshold is labeled as malicious traffic. This method is capable of detecting flooding attacks, where the sequence frequency goes beyond the threshold.
- **Attack severity identification:** Setting threshold values for different severity levels, with different occurrence frequencies for specific sequences.

5.8.1 Expert System Functions Required for Category 4 Detection

Category 4 detection is a combination of Category 2 and Category 3 detection mixed with state and statistics facts as its network context. To recognize Category 4 attacks, expert system rules describing it check the frequency of sequence. The context fact in Category 4 detection is *"State fact"* where the state number, state connection parameters and timestamp of the event occurrence are included, as well as "*Statistics fact*" where the timestamp of the event occurrence is included. The rule condition in Category 3 has *"State Criteria"* as its context criteria to check on the state facts, and "*Statistics Criteria*" as its context criteria to check on the statistics facts.

The functions required for Category 4 attacks are listed in the order of how a final attack state is detected as follows:

1. **Packet Searching C-Rule Base:** Packet Searches C-Rules, finding the first packet of the state transition
2. **Modify State Facts (Optional):** Modify state facts in pre-join block
3. **Pre-joining in Pre-join Block for States Checking (Optional):** Check if the state criteria of any rules are satisfied
4. **Activate New C-Rule:** Activating a new C-Rule that targeting to a new state. The C-rule embeds with the state's context information.
5. **Invalidate C-Rules** if final state has been reached**:** Invalidate any C-Rules related to this connection
6. **Repeat Step 1-5** if the attack doesn't reach the state that is counted in the statistics or

7. **Modifying Statistics Facts** if the state that is counted in the statistics has been reached.
8. **Pre-joining in Pre-join Block for Statistics Checking:** Adding or deleting a new time stamp of the event; Moving time window and Updating Statistics; Checking statistics value with threshold in rules;
9. **Repeat 1-9** when more packets in the flooding come if the threshold hasn't been reached
10. **Dropping Packet and Sending System Alert:** Drop any more flooding packets and send an alert message to the system.

Step 2 and 3 can be skipped when a direct activation from a C-Rule is valid. Usually in a state transition driven only by a network packet, a direct activation from a C-Rule can be executed. This is because when this C-Rule is matched, the activated C-Rule's context criterion is satisfied. When the state transition requires not only a network packet, but also other network context, a match of the first C-Rule couldn't directly activate new C-Rules. Prejoining is required to check if other network context criteria are satisfied or not.

Therefore, seven functions are required in Category 4 attacks. They are "Searching C-Rule base", "Modifying State Facts", "Pre-joining for State Transition", "Activating C-Rules", "Modifying Statistics Facts", "Pre-joining for Statistics Updating" and "Dropping Packets and Sending System Alert". Details of these functions have been covered in previous Sections 5.4.1, 5.6.3, 5.6.4, 5.4.3, 5.5.3, 5.5.4, and 5.4.2 respectively.

5.8.2 Category 4 Simulation and Performance Analysis

For Category 4, three attacks are chosen to simulate the system. They are Neptune, Portsweep and Processtable attack. Simulation results are shown in Table 5.6, while the performance analysis is shown in Table 5.7.

Neptune attack is a flooding of half open connections on a TCP server. Because of its connection based context, Neptune attack leads to a great amount of activation in the system (360, 3.0% of searches) and many writing cycles (1080) to the CAM. The required C-Rule per activation is only 1 because of the simple TCP transition. The required writing cycle per C-Rule is 3 clock cycles. The maximum CAM entries required is large (42), since Neptune has a lot of half open connections that each needs to be maintained by a C-Rule in CAM.

Table 5.6 Simulation Results for Category 4 Detection

Category 4 Detection Result		Neptune		PortSweep		ProcessTable	
Input Traffic	Packets Examined	8330	2001	2016	2001	537	2001
	Real Attack Packets Number	40	0	90	0	147	0
	Time Duration	6.5s	207.7s	262.7s	207.7s	50.0s	207.7s
Search, Match	Search Number	11844	56684	2076	56684	2425	56684
	Matched Number	639	372	180	372	147	0
	Matched of Activated C-Rules	40	248	178	248	146	0
Activations	Activation Times	360	248	180	248	147	0
	Activation C-Rules	360	248	268	248	147	0
	CAM Writing Cycles	1080	744	1156	744	441	0
In-validate	Invalidation Times	280	124	1	124	147	0
	In-validation C-Rules	280	124	1	124	147	0
C-Rule	C-Rule Number in C-Rule Base	42	5	92	5	2	1
System Speed	Slow Updates	0	0	0	0	0	0

Table 5.7 Performance Analysis for Category 4 Detection

	Neptune and Portsweep Attack		ProcessTable Attack	
	During attack	**Clean Traffic**	**During attack**	**Clean Traffic**
Number of Updates of C-Rules	3.0% - 8.7% of number of searches in time window	0.43% of number of searches in time window	6.1% of number of searches in time window	0
Number of C-rule Generation Per Update	1-2	1	1	1
Time of C-Rule Update Per Update	3~6 CAM cycles (30 - 60 ns)		3 CAM cycles (30 ns)	
Total Number of C-Rules in CAM	Real Flooding packets in time window ~100	~0.009% of number of searches in time window	2	1

With clean traffic, Neptune rules are matched most often by normal TCP connections. Since no flooding attacks are involved, the activation number goes down to normal TCP connection level (0.44%). C-Rule number per activation and writing cycles per C-Rule are exactly the same with the previous column since only the TCP state transition is involved. The CAM space required is small (0.009% of searches) as not many TCP connection need to be maintained at one time.

Portsweep is a flooding of SYN packet sending to different destination in order to probe their availability. As with IPsweep, we need a first C-Rule that has a specific source IP but no specific destination IP in order to count the different destinations that have been probed. We use Portsweep detection rules together with TCP connection rules in this simulation. Since it has a connection based context, this attack file generates very high number of activation (180, 8.7% of searches). The C-Rule per activation is 1.5, and writing cycles per C-Rule is 4.3 to write both TCP connection transition and attack transition. Since one C-Rule per probing needs to be maintained in C-Rules in order to find different probe attempts, the CAM size required for this attack is large (92 CAM entries).

Clean data has no Portsweep attack. Therefore the activation performance with Portsweep rules is the same as clean traffic with Neptune detection rules. Only TCP connection triggers activations.

Processtable is a flooding of established TCP connection on port 22 to make the server unavailable to other task. Due to its connection based context, it has a high activation number (147, 6.1% of searches) and the same performance in the aspect of C-Rule number per activations (1) and writing cycles per C-Rule (3), as Neptune detection. This is because they all uses TCP connection rules, and the packets triggering the rules are all from TCP connections. The CAM size required is small (2) since these TCP connection are established and finished, therefore it invalidates the C-Rules right away.

In clean traffic, there is no TCP connection established on port 22. Therefore no activation occurs with the clean traffic file for Processtable detection.

In conclusion, Category 4 detection usually requires high activation, high writing cycles and variable CAM space. If the traffic contains many open transitions or probing attempts which will require C-Rules to be maintained, the CAM space required is large. Otherwise, the CAM space requirement is low and can easily be satisfied. The average C-Rule number per activation is 1 to 1.5, and the average writing cycles per C-Rule are 3 to 4.3. Due to its flooding nature, the CAM match number is higher than in Category 3 detection. The updating time in Category 4 is short enough to keep up with the wire speed detection for Gigabit networks.

5.8.3 Category 4 Detection Examples: Neptune (Syn Flooding) Attacks

C-Rules for category 4 will be mapped into CAIPES. These rules are related to the context (e.g., network status and the statistics collected from previous packets). We use Syn flooding attack (named Neptune in Lincoln Lab data) as an example to show C-Rules for the category 4 expert rules within different network contexts. C-Rules for the category 4 expert rules for different network contexts are shown in Figures 5.58 to 5.66.

Syn flooding attack detection will count the number of occurrences of state 2 (pending connection) at a point in time. Once the number of pending connections exceeds the threshold, it means the buffer used by the destination server to keep a record of pending connections has been exhausted. This causes buffer overflow. Therefore, when a TCP connection goes into state 2, we increment the counter. When an acknowledge packet comes in later to transit the system to state 3 (full connection), we decrement the counter. The counter statistics is not related to timing information. It shows the half open connection number happened to all ports of a server at one point in time.

In each rule, there are two parts: the condition part and the action part, that are shown in the two big gray blocks in Figure 5.58.

First C-rule of Neptune Detection: Establish a new state 1 for Neptune attack

```
Define_Rule Cat4_rule_1 "an example of Npetune attack rule --first rule"
  Variables ($src, $dst, $src_port, $dst_port,  $time, $seq_number)
  Fact_pointer (*state1)
  IF {
     Packet (protocol== TCP; TCP_flag= = SYN;              //Packet criteria on Packet fact
         src_IP==$src; dst_IP==$dst;                       //Interfactual criteria between Packet fact and State fact
         src_port = = $src_port; dst_port = =$dst_port;
         seq = = $seq_number;   timestamp= =$time;)
     ! State (attack_ID= =Neptune;                         //State criteria on state fact
          context_para= =$src $dst $src_port $dst_port; )  //Inter-factual criteria
   }
  THEN {
     Modify (Output (alert= 0; log_packet= 0; action= 0;))   //Modify alert fact to "pass" packet
     Assert (*state1<-State(attack_ID=Neptune; state_number=1;   //Add new state fact
                          context_para= $src $dst $src_port $dst_port;
                          seq = $seq_numbr; timestamp = $time;))
     Activate Cat4_rule_2;                                 //Rule activation and invalidation
     Related_Rule (Cat4_rule_2, Cat4_rule_4)               //Related rules in Pre-Join block
     Related_Fact (*state1)                                //Related facts in Pre-Join block
  }
```

Figure 5.58 The First C-Rule in Category 4 Detection to Detect Neptune Attack

C-rule 1 detects the first packet of newly established state transitions where there is no related context existing in Pre-join block. Using the priority search ability of CAM, the function usually handled in Pre-join block can be implemented in CAM. C-Rule 1's criteria can be put in a CAM entry right after other context-existing C-Rule, with its Context parameter being "X (don't care)". A match of C-Rule 1 in the CAM implies that there is no match with other C-Rules which have context information in them. This implication satisfies C-Rule 1's context that no previous facts related to this parameter was ever recorded in the system during this period of time.

The action when C-Rule 1 is matched is to add a fact into the fact base in the Pre-join block to update the statistics. As shown in Figure 5.59, the addition of this fact will also trigger the Pre-join block to find a match with Pre-join criteria in C-Rule 2 or C-Rule 4 depending on the current statistics context. In fact, the system can activate a new C-rule 2 or C-Rule 4 directly and write it in the C-Rule base. C-Rule 2 and C-Rule 4 have the same content for CAM, but different actions in RAM. A match of C-Rule 1 involves functions of "search", and "activate new C-Rules" and "check statistics".

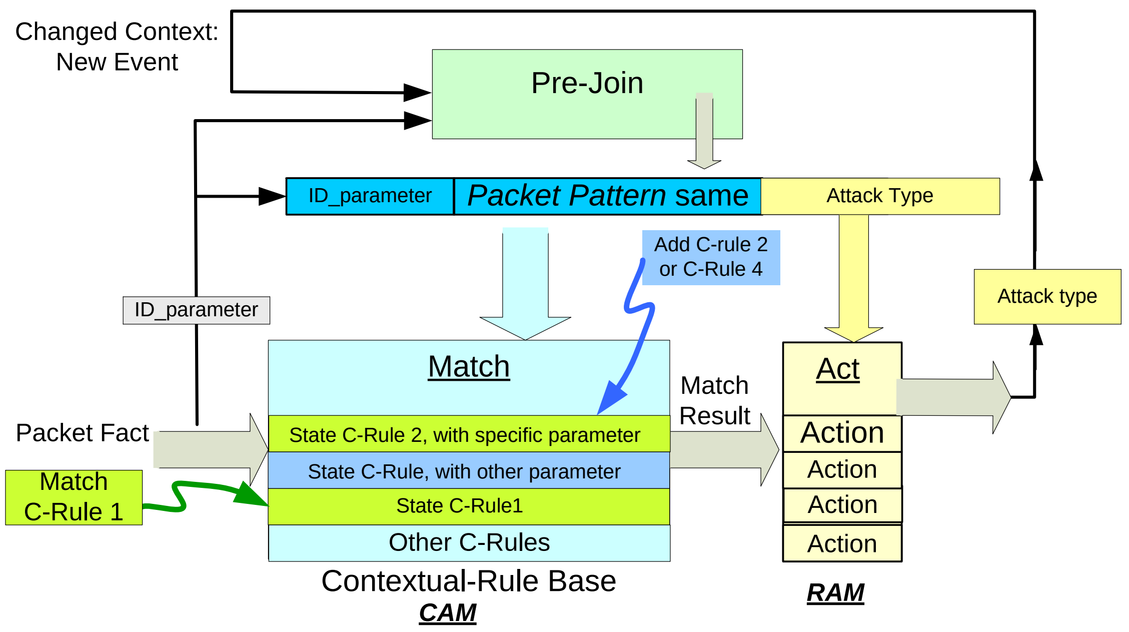


Figure 5.59 CAIPES After Match of C-Rule 1 in Category 4 Detection

Second C-rule of Neptune Detection: Transition from state 1 to state 2, counter increment

```
Define_Rule Cat4_rule_2 “an example of Neptune attack rule --second rule”
  Variables ($src, $dst, $src_port, $dst_port,  $time, $seq, $new_seq )
  Fact_pointer (*state1, *stat1)
  IF {
      Packet (protocol== TCP; TCP_flag= = SYN+ACK;            //Packet criteria on Packet fact
           src_IP==$dst; dst_IP==$src;                 //Interfactual criteria between Packet fact and State fact
           src_port = = $dst_port; dst_port = =$src_port;
           seq = = $new_seq; ack = = $seq_number+1; timestamp= =$time;)
       *state1 <- State (attack_ID= =neptune;    state_number==1;          //State criteria on state fact
                    context_para= =$src $dst $src_port $dst_port;          //Inter-factual criteria
                    seq= = $seq_number;  )
       *stat1<-Statistics (attack_ID= =Neptune; stat_number <$threshold-1;   //Statistics criteria
                         context_para ==?src ?dst;)         //Interfactual criteria between packet and statistics
  }

  THEN {
      Modify (Output (alert= 0; log_packet= 0; action= 0;))          //Modify alert fact to “pass” packet
      Modify (*state1 (state_number=2; seq=$new_seq; timestamp = $time;)  //Modify the state fact
      Modify (*stat1 (stat_number;))                                //Modify the statistics fact
      Activate Cat4_rule_3;                                         //Rule activation and invalidation
      Related_Rule (Cat4_rule_3)                                    //Related rules in Pre-Join block
      Related_Fact (*state1, *stat1)                                //Related facts in Pre-Join block
  }
```

Figure 5.60 The Second C-Rule in Category 4 Detection to Detect Neptune Attack

In the conditions part of Category 4 rules, there are three types of conditions involved, separated by the fine white lines in the gray “condition” blocks. The first type of condition is “related criteria to packet fact”. This includes “packet criteria on packet fact” and “inter-factual criteria between packet fact and state fact”. The second type of condition is “related criteria to state fact”. This includes “state criteria on state fact” and “inter-factual criteria between state fact and packet fact". The third type of condition is "related criteria to statistics fact" which includes "statistics criteria on statistics fact" and "inter-factual criteria between statistics fact and packet fact".

We use fact pointers to store the pointer of specific facts. "State1" is a pointer, and "<-" sign assign the address of a state fact to it. When we use "state1" again in this rule, the specific state fact will be manipulated. "Stat1" is another pointer, and "<-" sign assign the address of a statistics fact to it. When we use "stat1" again in this rule, the specific statistics fact will be manipulated.

The action in C-Rule 2 includes an increment to the counter that counts half open connections. As shown in Figure 5.61, when C-Rule 2 matches against the current packet, the system will trigger the Pre-join block to activate a new C-Rule 3. In fact, the system can activate a new C-rule 3 directly and write it in the C-Rule base. This C-Rule 3 will overwrite the C-Rule 2 in the C-Rule base. A match of C-Rule 2 involves functions of "search", and "activate new C-Rules", "modify statistics", "pre-joining with statistics updates",

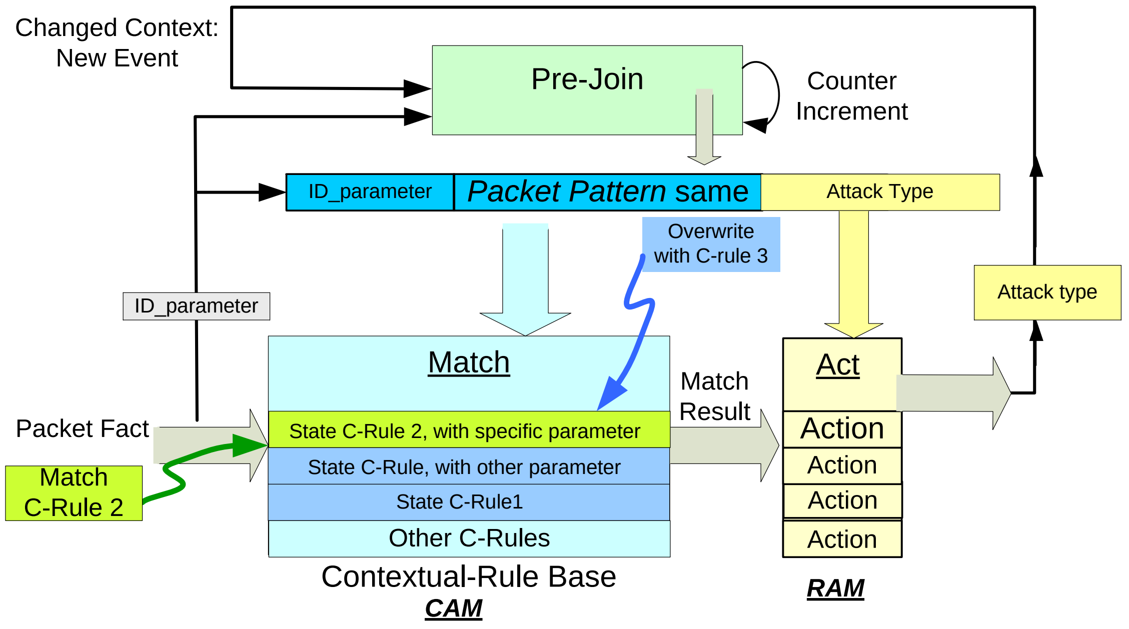


Figure 5.61 CAIPES After Match of C-Rule 2 in Category 4 Detection

Third C-rule of Neptune Detection: Transition from state 2 to state 3, counter decrement

```
Define_Rule Cat4_rule_3 “an example of Neptune attack rule --third rule”
  Variables ($src, $dst, $src_port, $dst_port,  $time, $threshold, $seq, $new_seq )
  Fact_pointer (*state1, *stat1)
  IF {
      Packet (protocol== TCP; TCP_flag= = ACK;              //Packet criteria on Packet fact
          src_IP==$src; dst_IP==$dst;                   //Interfactual criteria between Packet fact and State fact
          src_port = = $src_port; dst_port = =$dst_port;
          seq = = $new_seq; ack = = $seq_number+1; timestamp= =$time;)
       *state1 <- State (attack_ID= =neptune;    state_number==2;              //State criteria on state fact
                  context_para= =$src $dst $src_port $dst_port;              //Inter-factual criteria
                  seq= = $seq_number;  )
       *stat1<-Statistics (attack_ID= =Neptune; stat_number >0;   //Statistics criteria
                      context_para ==?src ?dst;)          //Interfactual criteria between packet and statistics
  }

  THEN {
      Modify (Output (alert= 0; log_packet= 0; action= 0;))          //Modify alert fact to “pass” packet
      Retract (*state1)                                     //Modify the state fact
      Modify (*stat1 (stat_number--;))                       //Modify the statistics fact
      Invalidate Cat4_rule_3;                              //Rule activation and invalidation
  }
```

Figure 5.62 The Third C-Rule in Category 4 Detection to Detect Neptune Attack

A match of C-Rule 3 means the TCP connection is fully established. As shown in Figure 5.63, the Pre-join block will then delete this C-Rule because it is already the final state of the TCP connection. A match of this C-Rule also invalidates one half-open connection, thus we decrease the counter by 1. A match of C-Rule 3 involves functions of "search", and "Invalidate C-Rules", "modify statistics", "pre-joining with statistics updates", and "system output".

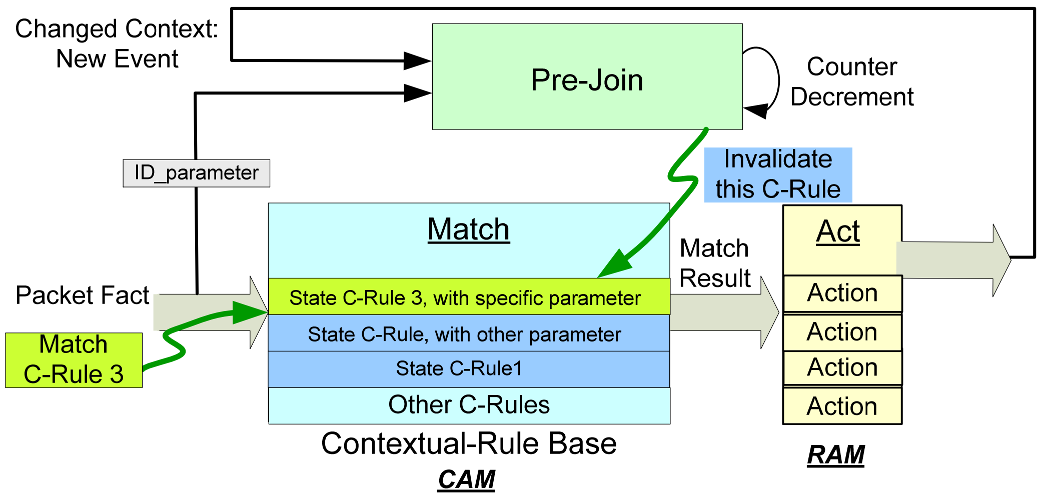


Figure 5.63 CAIPES After Match of C-Rule 3 in Category 4 Detection

Fourth C-Rule of Neptune Detection: Flooding of half open connection

```
Define_Rule Cat4_rule_4 "an example of Neptune attack rule --fourth rule"
  Variables ($src, $dst, $src_port, $dst_port,  $time, $threshold, $seq, $new_seq )
  Fact_pointer (*state1, *stat1)
  IF {
      Packet (protocol== TCP; TCP_flag= = SYN+ACK;            //Packet criteria on Packet fact
          src_IP==$dst; dst_IP==$src;                          //Interfactual criteria between Packet fact and State fact
          src_port = = $dst_port; dst_port = =$src_port;
          seq = = $new_seq; ack = = $seq_number+1; timestamp= =$time;)
       *state1 <- State (attack_ID= =neptune;    state_number==1;          //State criteria on state fact
                  context_para= =$src $dst $src_port $dst_port;            //Inter-factual criteria
                  seq= = $seq_number;  )
       *stat1<-Statistics (attack_ID= =Neptune; stat_number = =$threshold-1;   //Statistics criteria
                      context_para ==?src ?dst;)         //Interfactual criteria between packet and statistics
  }

  THEN {
      Print (Attack ID: Neptune, Flooding of half open connections, Dropped")   //print alert message
      Modify (Output (alert= 1; log_packet= 1; action= 2;))           //Modify alert fact to "pass" packet
       Modify (*state1 (state_number=2; seq=$new_seq; timestamp = $time;)  //Modify the state fact
      Modify (*stat1 (stat_number++;))                                //Modify the statistics fact
      Activate Cat4_rule_3;                                           //Rule activation and invalidation
      Related_Rule (Cat4_rule_3)                                      //Related rules in Pre-Join block
      Related_Fact (*state1, *stat1)                                  //Related facts in Pre-Join block
  }
```

Figure 5.64 The Third C-Rule in Category 4 Detection to Detect Neptune Attack

A match of C-Rule 4 means the threshold of half open connection has been reached. Therefore the system gives an alert message and drops the packet. As shown in Figure 5.65, it also activates C-Rule 3 in case it later turns out to be a fully established connection. C-Rule 3 will overwrite C-Rule 4. A match of C-Rule 4 involves functions of "search", and "modify statistics", "pre-joining with statistics updates", and "activate new C-Rules".

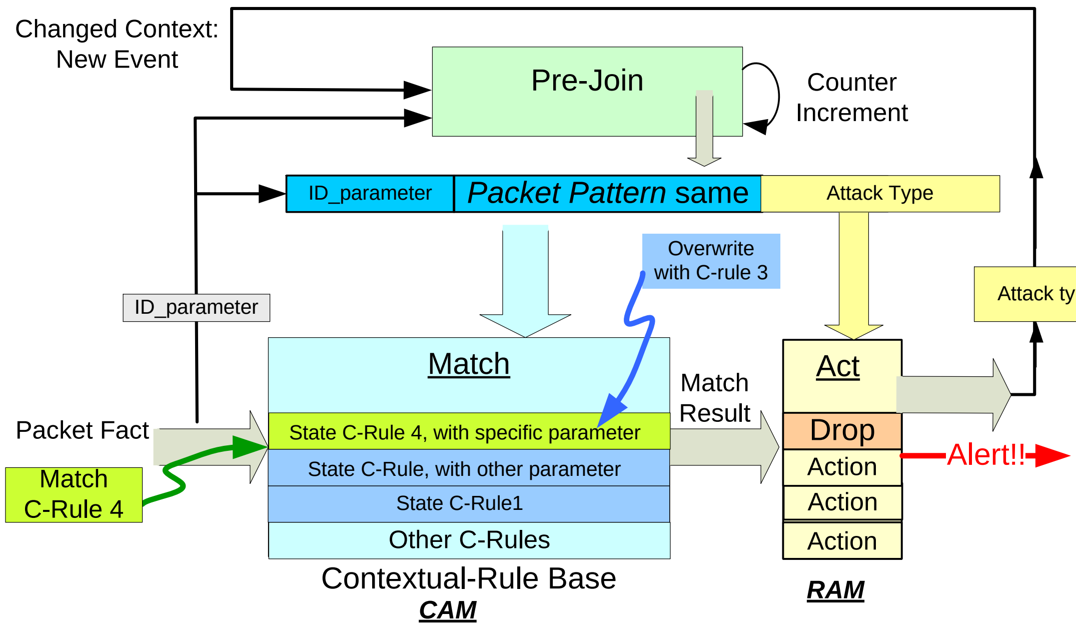


Figure 5.65 CAIPES After Match of C-Rule 4 in Category 4 Detection

Clean-up Rule in Pre-join Block: Delete the fact and the C-Rule when the server sends a reset packet to reset the half-open connection

```
Define_Rule Cat4_rule_4 “an example of Neptune attack rule --fourth rule”
  Variables ($src, $dst, $src_port, $dst_port,  $time, $threshold, $seq, $new_seq )
  Fact_pointer (*state1, *stat1)
 IF {
     Packet (protocol== TCP; TCP_flag= = RESET;           //Packet criteria on Packet fact
          src_IP==$dst; dst_IP==$src;                     //Interfactual criteria between Packet fact and State fact
          src_port = = $dst_port; dst_port = =$src_port; timestamp= =$time;)
      *state1 <- State (attack_ID= =neptune;    state_number==2;        //State criteria on state fact
                  context_para= =$src $dst $src_port $dst_port;)         //Inter-factual criteria

      *stat1<-Statistics (attack_ID= =Neptune;                           //Statistics criteria
                      context_para ==?src ?dst;)          //Interfactual criteria between packet and statistics
 }
 THEN {
     Modify (Output (alert= 0; log_packet= 0; action= 0;))        //Modify alert fact to “pass” packet
     Modify (*stat1 (stat_number--;))                              //Modify the statistics fact
     Retract (*state1)                                             //Delete the state fact
     Invalidate Cat4_rule_2;                                       //Delete a C-Rule
  }
```

Figure 5.66 The Cleanup C-Rule in Category 4 Detection to Detect Neptune Attack

The clean-up rule is stored in the C-rule base. Theoretically the reset packet can happen after any state to reset the TCP connection, but in reality only a small percentage of them (1%) indeed occur. This leads to inefficiency when making the cleanup C-Rule a possible transition from each state with different contexts. Therefore, we use only one “clean up C-Rule” with all its inter-factual criteria Context parameter as “don't care” bits. This greatly reduces the inefficient C-Rule activations and also saves C-Rule base space. AS shown in Figure 5.67, when the clean-up rule is matched, we will invalidate the current TCP state. If the TCP state is 2, which is a TCP half open connection, the counter in the fact base will also have a decrement. The functions involves in the match of the clean-up rule includes "modify statistics", "pre-joining with statistics updates", and "invalidate C-Rules".

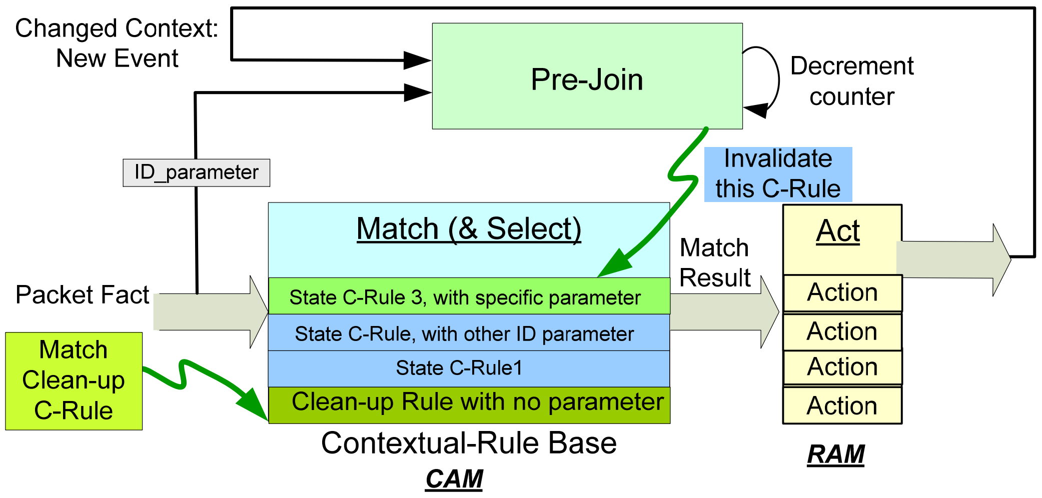


Figure 5.67 CAIPES After Match of Clean-up C-Rule in Category 4 Detection

The features of the C-Rules of this State Transition Tracking architecture are as follows:

- Every new match of a C-Rule in Rule Base will cause the Pre-join to generate new C-Rules leading to the next state.
- Old C-Rules which will never generate matches will be deleted or overwritten in time.

5.8.4 Category 4 Detection Conclusion

Category 4 attack can be implemented in C-Rule method with CAM and RAM. It requires moderately high numbers of CAM modifications and each modification requires 3-5 writing cycles. Detection towards three attacks from Lincoln Lab data has been implemented in a prototype system and it can keep up with gigabit networks.

5.9 PROTECTION AGAINST CAM BUFFER OVERFLOW

Because of the limited physical space of CAM, CAIPES will have a buffer overflow risk under certain circumstances when attackers are deliberately targeting the CAIPES system or the target server. Buffer overflow is caused by CAIPES' activating too many C-Rules which fill up the CAM. Therefore CAIPES doesn't know what to do and could thus not handle any more traffic.

A straightforward method is used here as shown in Figure 5.68. We check the water level of the CAM entry usage. Once it reaches each level's limit, CAIPES starts stopping specific packets to slow down the C-Rule addition and protect against CAM buffer overflow. Once it reaches the final level, the CAIPES system will stop adding any C-Rules and offload the traffic to a processor.

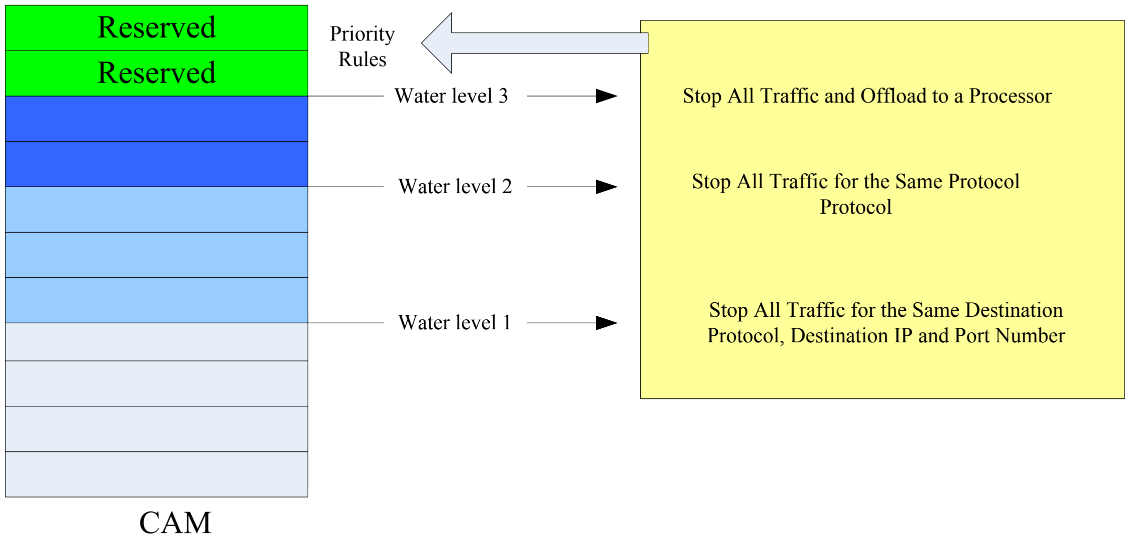

Reserved
Reserved
Priority Rules
Water level 3
Water level 2
Water level 1
Stop All Traffic and Offload to a Processor
Stop All Traffic for the Same Protocol Protocol
Stop All Traffic for the Same Destination Protocol, Destination IP and Port Number
CAM

Figure 5.68 Three Water Levels in CAM and Graceful Degradation to Protect CAM Buffer Overflow

There are three levels of water level associated with graceful degradation for the traffic management. The first one happens when the allocated space for an attack is filled up. We stop all traffic destined to the same target in case it is a DOS attack to that computer. If the CAM keeps activating new C-Rules and the water level reaches level 2 where the free space for any attack in CAM was filled up by 60%, we stop all traffic for that same protocol. The final level is when the CAM is filled up except the "reserved entry", we stop all traffic and send them to an offload processor. This protects the CAIPES from buffer overflow while the traffic is still getting inspected in CAIPES hardware to the largest extent possible according to CAM's ability.

5.10 CONCLUSION

This chapter presented a unified IPS expert system architecture that uses a highly parallel search memory CAM to perform intrusion detection and prevention in Gigabit networks at wire speed. Five major contributions are:

1. A novel Contextual-Rule method is invented to convert the two-database match problem of expert systems into a one-database search problem. The expert system functions remain the same for IDS/IPS domain.
2. Parallel search engine CAM has been applied to the Contextual-Rule base to perform the match in parallel, thereby accelerating the expert systems for network IPS application.
3. This unified CAM-Assisted Intrusion Prevention Expert System (CAIPES) architecture supports detection methods towards various multi-packet attacks, including flooding of packet attacks, multi-packet sequence attack, flooding of sequence attack. It also provides prevention actions including stopping malicious traffic, allowing good traffic, sending suspicious traffic to lower priority queues, and logging packets and alerting users
4. A prototype of CAIPES has been developed. Nine detection methods from the three multi-packet attacks have been implemented in the prototype system. We use traffic sniffing data from Lincoln Lab IDS evaluation dataset as the simulation data. A total of 4 MB of attack traffic data and 7 MB of clean traffic data have been run through the simulation. Simulation shows CAIPES has the performance necessary to keep up with Gigabit detection.
5. Scalability of this CAIPES architecture has been discussed and a protection mechanism with graceful degradation to protect CAM from buffer overflow.

6.0 CONCLUSIONS

Expert systems are one existing IDS method that is functionally effective in detecting network intrusions. However, in general they are computationally intensive and usually too slow for high speed or real-time demands. Their performance is insufficient to keep up with gigabit network traffic. This thesis has demonstrated that Content Addressable Memory (CAM) can be used to accelerate IDS/IPS expert systems. This chapter offers a summary of our research findings.

6.1 CONTRIBUTION SUMMARY

Content Addressable Memory (CAM) is a hardware parallel search engine chip that has been applied to accelerate a network Intrusion Prevention expert system in this thesis. The summary of the contributions of this thesis are as follows:

1. A novel encoding scheme for single-cycle CAM range mapping has been described in this thesis. Single-cycle CAM operation can finish range criteria matching in IDS/IPS rules. This encoding scheme is a tradeoff between CAM width and CAM depth comparing to existing codes of binary code, one-hot encoding, two-hot encoding, gray coding and Johnson coding. This is the foundation of mapping IDS/IPS rules as they usually contain range matching criteria.
2. A method to map detection rules of single-packet attack into CAM has been shown in this thesis. Detection ability includes header checking, range matching, short payload string matching, long string matching, multiple string matching, and payload string matching with distance ranges between them. Novel mapping architecture of single-packet detection rules, which is based on the range matching encoding scheme from contribution 1, was shown in the thesis to solve 17.9%-28.4% of Snort rules. With

the novel mapping architecture, a total of up to 91% of an open source IDS Snort rules (version 2.0) are mapped to a CAM automatically using a Perl script. This is the foundation of multiple-packet detection in CAM.

3. A method to map detection rules of multi-packet attack into CAM has been shown in this thesis. Detection ability includes flooding packet attacks, sequence of packet attacks, and flooding of sequence attacks. Three attacks from each category were hand coded into a CAM and simulated with real network traffic. This mapping of multiple-packet detection uses range matching encoding scheme from contribution 1, as well as on the foundation of CAM based single-packet detection from contribution 2.
4. A Contextual Rule (C-Rule) method that pre-joins part of the fact base and the rule base has been introduced in this thesis for expert system acceleration. This converts the two-database matching problem of an expert system into a one-database (Contextual Rule base) problem. Thus, the parallel searching capability of CAM can be applied directly for acceleration. This general architecture is developed from contribution 3 and may be extended to accelerate other expert system applications of similar structure. The application limitations of this C-Rule method are discussed in Section 6.3.
5. A prototype system with a SystemC CAM and RAM behavioral models is implemented. Traffic sniffing data from Lincoln Lab Intrusion Detection Evaluation Dataset is used here. Packet files with specific attacks involved, as well as attack free packets files are simulated through the prototype.

6.2 SIGNIFICANCE OF THIS RESEARCH

This research demonstrated CAM's capability in accelerating network IDS, especially in multi-packet detection IDS. Previously, CAM has only been used to accelerate single-packet detection IDS/IPS such as Snort.

This thesis also improves the CAM-based single-packet detection IDS. It first showed a novel encoding scheme for single-cycle range matching in CAM, as a new method to implement the “range criteria” in IDS rules without adding extra matching cycles. Based on this method, new architecture with CAM has been designed for complicated single-packet detection IDS.

The more significant contribution concerns using CAM on multiple-packet detection IDS/IPS with expert system rules. A new Contextual Rule method has been introduced for IDS/IPS expert system to use CAM’s parallel search power. Therefore, different kinds of multiple-packet detection can be mapped to CAM using Contextual Rule. This method provides a general architecture of how to map expert systems with two-database matching into a one-database search engine CAM. It also provides a new method of mapping multiple-packet detection rules into a CAM.

6.3 LIMITATION OF THE RESEARCH

When applying this research to a broader area, there are some limitations. For the IDS/IPS problem domains, CAIPES’ application is limited to certain type of detection because of the computational limitation of CAM, and the C-Rule method performance limitation. For other expert system problem domains, such as medical diagnosis system or traffic control system, C-Rules can only be applied to limited types of problem domains. The limitations are detailed in the following section.

First, CAM has a computational limitation. It is good at exact matching but is not good at other complicated computations such as mathematical expressions. Some experienced attackers are using stealthy intrusions for which a complicated mathematical model would be required to detect the traffic abnormality over a long period of time. CAM has very limited capability to handle this type of detection.

Also, detection of some attacks requires too many C-Rule activations and invalidations from the C-Rule base, thereby degrading CAIPES performance to an unacceptable degree. Some anomaly detection systems define good traffic patterns and drop unmatched traffic. In this case, CAIPES may cause too many C-Rule activations in the C-Rule base and risk spending too much time updating C-Rules. Thus, it could be unable to handle real time traffic at wire speed. Therefore, CAIPES is good at misuse detection where CAIPES stores rules for bad traffic. In this case, only a small percentage of traffic will trigger the C-Rule updating. Thus, the updating can be finished during two packets' interval. CAIPES is not good at anomaly detection where CAIPES stores rules for good traffic since these rules will be matched and activate new rules very often.

If we want to apply the C-Rule method to other expert systems so that they can also use CAM to accelerate the execution, there is also a limitation of the applications. The C-Rule method is based on a specific feature of expert system "facts" such that all facts in that expert system are of two types. One fact type is the driven fact that would cause the expert system to execute and generate an output result (e.g. packet fact in IDS/IPS problem domain). The other fact type is the environment fact that alone do not cause the expert system to execute, but together with the driven fact, will cuase the expert system to generate a result based on both facts (e.g., the intermediate state of a TCP transition). Therefore, if an expert system has the two types of facts mentioned above, the C-Rule method can always make the first type of fact searching a CAM, with the second type of facts pre-joining with rules to create C-Rules. If an expert system has more than one type of driven fact, the C-Rule method cannot accommodate them into one vector to search the CAM. The variable binding requirements between these facts are not solved using the C-Rule method.

Therefore, the C-Rule method application can only be applied to problem domains for expert systems where the driven facts of this expert system are of one type.

6.4 FUTURE DIRECTION

Our future directions include but are not limited to the following list:

1. Explore the possibility of mapping PERL regular expression string matching into CAM. This would become a general CAM solution for string matching problem, not only for single-packet detection, but also for file searching and other applications.
2. Explore the C-Rule performance with different attacks. Currently, classification of attacks is based on expert system detection ability. Three attacks from each category demonstrate the detection ability of CAIPES. These attacks also show how different attacks have different influences on C-Rule performance. In order to further analyze the C-Rule method performance, in the future the 52 types of Lincoln Lab attacks can be classified according

to their impacts on the performance of the C-Rule method. Choosing typical attacks from each category to analyze C-Rule performance would evaluate whether the C-Rule activation can be fast enough for all attacks to keep up with the wire speed detection in multi-gigabit networks.

3. Explore the possibility of applying this C-Rule method into other expert system problem domains that have a single type of fact to drive the system. For certain expert systems, this method could convert the two-database search problem into a one-database search problem, and thus utilizing CAM’s parallel search power to accelerate the expert system matching process.

BIBLIOGRAPHY

[1] R. Heady, G. Luger, A. Maccabe, and M. Servilla. The Architecture of a Network Level Intrusion Detection System. Technical report, University of New Mexico, Department of Computer Science, Aug. 1990

[2] Vigna, G.; Kemmerer, R.A. "NetSTAT: a network-based intrusion detection approach," Computer Security Applications Conference, 1998, Proceedings., 14th Annual , 7-11 Dec 1998 pp25 -34

[3] T.Wan, X. Yang, "Intrudetector: a software platform for testing network intrusion detection algorithms," Proceedings of the 17th annual Computer Security Applications Conference, 2001. pp3-11

[4] Paulson, L.D.; "Stopping intruders outside the gates", Computer, Volume: 35 Issue: 11 , Nov. 2002, pp20 -22

[5] Sherif, J.S.; Dearmond, T.G. "Intrusion detection: systems and models," Enabling Technologies: Infrastructure for Collaborative Enterprises, 2002. Proceedings. Eleventh IEEE International Workshops, 2002, pp115 -133

[6] J. Allen, A. Christie, et al., "State of the Practice of Intrusion Detection Technologies," Technical Report CMU/SEI-99-TR-028, 2000, Carnegie Mellon University, Software Engineering Institute.

[7] John McHugh, "Intrusion and Intrusion Detection," CERT Coordinaton Center, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA, Published online: 2001. Springer Verlag, http://www.cs.plu.edu/pub/faculty/spillman/seniorprojarts/ids/iids.pdf

[8] Reuters, "Virus damage estimated at $55 billion in 2003", Jan. 16, 2004 http://msnbc.msn.com/id/3979687

[9] Computer Economics www.computereconomics.com.

[10] Richard Pethia, "Attacks on the Internet in 2003", Global Issues, Volume 8, Number 3, November 2003 http://usinfo.state.gov/journals/itgic/1103/ijge/gj11a.htm

[11] CSI/FBI Computer Crime and Security Survey SI Computer Security Issues & Trends 2002

[12] Stephen Northcutt, Judy Novak, “Network Intrusion Detection,” 3rd Edition, Sept. 2002,New Riders Publishing

[14] Rebecca Gurley Bace, “Intrusion Detection,” Mcmillan Technical Publishing, 2000

[15] Peng Ning, Ph.D thesis, “Abstraction-based Intrusion Detection in Distributed Environments,” George Mason University, 2001.

[16] Kemmerer, R.A.and Vigna, G., “Intrusion detection: a brief history and overview,” Computer, Volume: 35 Issue: 4, Apr 2002, pp27 –30

[17] M. Esmaili,et al. “Case-Based Reasoning for Intrusion Detection”, 12th Annual Computer Security Applications Conference, December 09 - 13, 1996, San Diego California, pp214-222

[18] Rebecca Bace and Peter Mell, “Intrusion Detection Systems”, NIST Special Publication on Intrusion Detection System，http://csrc.nist.gov/publications/nistpubs/800-31/sp800-31.pdf

[19] John Mchugh, Alan Christie, Julia Allen, “Defending Yourself: The Role of Intrusion Detection Systems”, IEEE Software, Vol 17, Issue 5, September 2000 pp.42-51

[20] Ilgun, K.; Kemmerer, R.A.; Porras, P.A. “State transition analysis: a rule-based intrusion detection approach,” Software Engineering, IEEE Transactions on , Volume: 21 Issue: 3 , Mar 1995. pp181 -199

[21] J. P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, James P Anderson Co., Fort Washington, PA, apr 1980.

[22] Dorothy E. Denning, “An Intrusion-Detection Model, IEEE TRANS. Software Engineering, Vol. SE-13, NO. 2, FEB 1987, pp222-232.

[23] Heberlein, T., K. Levitt and B. Mukherjee. “A Method to Detect Intrusive Activity in a Networked Environment.” Proceedings of the 14th National Computer Security Conference, 1991. pp362-372

[24] Heberlien, T., B. Mukherjee, K.N. Levitt; G. Dias and D.Mansur. “Towards Detecting Intrusions in a Networked Environment.” Proceedings of the Fourteenth Department of Energy Computer Security Group Conference, 1991. pp47-66

[25] Mukherjee, B.; Heberlein, L. T. and K.N Levitt. “Network Intrusion Detection,” *IEEE Network*, volumn8, issues3, pp26-41, 1994.

[26] Snapp, S., et al, “A System for Distributed Intrusion Detection.” Proceedings of COMPCON Spring ’91,San Francisco, CA, 1991.

[27] Crosbie, M. and E. Spafford. Defending a Computer System Using Autonomous Agents.” *Proceedings of the Eighteenth National Information Systems Security Conference*, Baltimore, MD, 1995.

[28] Yuebin Bai and Hidetsune Kobayashi, "Intrusion Detection Systems: Technology and Development," Advanced Information Networking and Applications, AINA 2003. 17th International Conference, 27-29 March 2003, pp 710 –715

[29] Yuebin Bai; Kobayashi, H., "New string matching technology for network security," Advanced Information Networking and Applications, 2003. AINA 2003. 17th International Conference, 27-29 March 2003, pp198 -201

[30] Kruegel, C.; Valeur, F.; Vigna, G.; Kemmerer, R. "Stateful intrusion detection for high-speed networks," Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on , 12-15 May 2002, pp266 –274

[31] Teresa F. Lunt, R. Jagannathan, Rosanna Lee, Sherry Listgarten, David L.Edwards, Peter G. Neumann, Harold S. Javitz, and Al Valdes. IDES: The enhanced prototype, A real-time intrusion detection system. Technical Report SRI Project 4185-010, SRI-CSL-88-12, CSL SRI International, Computer Science Laboratory, SRI Intl. 333 Ravenswood Ave., Menlo Park, CA 94925-3493, USA, October 1988.

[32] Debra Anderson, Teresa F. Lunt, Harold Javitz, Ann Tamaru, and Alfonso Valdes. Detecting unusual program behavior using the statistical component of the next-generation intrusion detection system (NIDES). Technical Report SRI-CSL-95-06, Computer Science Laboratory, SRI International, Menlo Park, CA, USA, May 1995

[33] Michael M Sebring, Eric Shellhouse, Mary E Hanna, and R Alan White-Hurst. Expert systems in intrusion detection: A case study. In Proceedings of the 11th National Computer Security Conference, pages 74-81, Baltimore, Maryland, October 17-20, 1988. National Institute of Standards and Technology/National Computer Security Center.

[34] Philip A Porras and Peter G Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In Proceedings of the 20th National Information Systems Security Conference, pages 353-365, Baltimore, Maryland, USA, October 7-10 1997. National Institute of Standards and Technology/National Computer Security Center.

[35] CMDS, www.saic.com

[36] S. Kumar, "Classification and Detection of Computer Intrusions," Ph.D Thesis, Purdue Unviersity, August, 1995

[37] S.Kumar and E.H.Spafford, "A Pattern matching model for misuse intrusion detection," Proceedings of the 17th National Computer Security Conference, pp11-21, October, 1994

[38] H S Vaccaro and G E Liepins. Detection of anomalous computer session activity. In Proceedings of the 1989 IEEE Symposium on Security and Privacy, pages 280-289, Oakland, California, May 13, 1989. IEEE Computer Society Press.

[39] K. Chen, S.C.Lu, and H.S. Teng, "Adaptive real-time anomaly detection using inductively generated sequential patterns," in Proc. IEEE Symp. Res. Security, Privacy, Oakland, CA, May 1990, pp278-295

[40] A. Sundaram, “An Introduction to Intrusion Detection,” 1996.

[41] D Anderson, T Frivold, and A Valdes. Next-generation intrusion-detection expert system (NIDES). Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, Menlo Park, CA 94025-3493, USA, May 1995.

[42] Peter Jackson, *Introduction to Expert Systems*, 3rd edition

[43] Joseph C. Giarratano, *Expert Systems: Principle and Programming*, 3rd edition

[44] U. Lindqvist and P. Porras. *eXpert-BSM: A host-based intrusion detection solution for Sun Solaris*. In Proc. of the 17 th Annual Computer Security Applications Conference, Dec. 2001

[45] *Lunt, T.F.; Jagannathan, R.; et al. Knowledge-based intrusion detection,* AI Systems in Government Conference, 1989.,Proceedings of the Annual , 27-31 March 1989 Pages:102 - 107

[46] Lindqvist, U.; Porras, P.A.; *Detecting computer and network misuse through the production-based expert system toolset (P-BEST),* Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on , 9-12 May 1999 Pages:146 - 161

[47] Lindqvist, U.; Porras, P.A *Detecting computer and network misuse through the production-based expert system toolset (P-BEST),* Slides, http://www.ce.chalmers.se/staff/ulfl/pubs/sp99lp-slides/SP99-PBEST-May99.PPT

[48] Phillip A. Porras, Peter G. Neumann, *EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances* Proc. 20th NIST-NCSC National Information Systems Security Conference

[49] M. Roesch, “Snort- Lightweight Intrusion Detection for Networks,” Proc. LISA’99: 13th System Administration Conf.*, Seattle, WA, Nov.1999.*

[50] “CLIPS: A Tool for Building Expert Systems”, http://www.ghg.net/clips/CLIPS.html

[51] S. J. Stolfo et al. PARULEL: Parallel rule processing using meta-rules for redaction. Journal of Parallel and Distributed Computing, 13:366--382, December 1991.

[52] A. Gupta, C. Forgy, A. Newell, High-speed implementations of rule-based systems, ACM Transactions on Computer Systems, Volume 7 , Issue 2 (May 1989) Pages: 119 – 146

[53] Srinivasa R., Parallel Rule-Based Isotach Systems, Technical Report CS-1999-04, Dept. of Computer Science, University of Virginia, Feb 1999

[54] A. Gupta et al. “Parallel Algorithms and Architectures for Rule-Based Systems”, Proceedings of the 13th annual international symposium on Computer architecture, Tokyo, Japan, 1986, Pages: 28 – 37

[55] Charles Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", Artificial Intelligence, 19, pp 17-37, 1982

[56] Daniel P. Miranker. TREAT: A better match algorithm for AI production systems. In Proceedings of AAAI 87 Conference on Artificial Intelligence, pages 42--47, August 1987

[57] K. Oflazer, "Parallel Execution of Production Systems" August, 1984 International Conference on Parallel Processing, IEEE.

[58] P. Nayak, A. Gupta, and P. Rosenbloom. Comparison of the Rete and Treat production matchers for SOAR (a summary). In Proceedings of National Conference on Artificial Intelligence, pages 693--698, August 1988.

[59] 58. S. J. Stolfo and D. P. Miranker. DADO: A parallel processor for expert systems. In Proceedings of International Conference on Parallel Processing, pages 74--82, April 1984.

[60] David Elliot Shaw. NON-VON's Applicability to Three AI Task Areas. International Joint Conference on Artificial Intelligence, 1985.

[61] T. Hanyu, K. Takeda and T. Higuchi, Design of a Multiple-Valued Rule-Programmable Matching VLSI Chip for Real-Time Rule-Based Systems, IEEE Int. Symp. on Multiple-Valued Logic. Vol.22 pp274-281, 1992

[62] T. Hanyu, S. Abe, M. Kameyama and T. Higuchi, "Highly-Safe Intelligent Vehicle Using a New Content-Addressable Memory", Intelligent Vehicles '94 Symposium, Proceedings of the , 24-26 Oct. 1994 Pages:467 - 472

[63] T. Hanyu, K. Takeda and T. Higuchi, "Rule-Programmable Multiple-Valued Matching VLSI Processor for Real-Time Rule-Based Systems", IEICE Trans. on Electron. Vol. E76-C No.3 pp472-479, 1993

[64] Michael A. Kelly, Rudolph E. Seviora: "A Multiprocessor Architecture for Production System" Matching. Sixth National Conference on Artificial Intelligence, July 13-17, 1987, Seattle, WA pp36-41

[65] Michael A. Kelly, Rudolph E. Seviora: An Evaluation of DRete on CUPlD for OPSS Matching. of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989 pp84-90

[66] Jose Nelson Amaral and Joydeep Ghosh, "Associative Memories Provide an Efficient Control Mechanism for a Parallel Production System Architecture". In 10th Congress of the Brazilian Microelectronics Society, pages 329-338, July 1995.

[67] J. N. Amaral and J. Ghosh, "An Associative Memory Architecture for Concurrent Production Systems", 1994 IEEE Int'l Conf. on Systems, Man and Cybernetics, San Antonio, Oct, 1994, pp.2219-2224.

[68] J. N. Amaral and J. Ghosh, "Speeding up Production Systems: From Concurrent Matching to Parallel Rule Firing," XIX Latin American Informatics Conference, Buenos Aieres, Aug. 1993, pp. 163-182.

[69] MIT Lincoln Lab: DARPA Intrusion Detection Evaluation, http://www.ll.mit.edu/IST/ideval/index.html

[70] Kris Kendall, "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems", Master's Thesis, Massachusetts Institute of Technology, 1998. http://www.ll.mit.edu/IST/ideval/pubs/1998/kkendall_thesis.pdf

[71] Joshua W. Haines, Richard P. Lippmann, David J. Fried, Eushiuan Tran, Steve Boswell, Marc A. Zissman, "1999 DARPA Intrusion Detection System Evaluation: Design and Procedures", MIT Lincoln Laboratory Technical Report, http://www.ll.mit.edu/IST/ideval/pubs/2001/TR-1062.pdf

[72] John Pescatore, Gartner Research, "Intrusion Detection should be a function, Not a product", CSO Analyst Report, http://www.csoonline.com/analyst/report1660.html

[73] Garcia, R.C.; Sadiku, M.N.O.; Cannady, J.D.; "WAID: wavelet analysis intrusion detection", Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposium on , Volume: 3 , Aug. 4-7, 2002. pp688 -691

[74] P.-Y. Lee and A. M. K. Cheng, ``HAL: A Faster Match Algorithm," IEEE Transactions on Knowledge and Data Engineering, Vol. 14, No. 5, pp. 1047-1058, September/October 2002.

[75] J. A. Kang and A. M. K. Cheng, ``Shortening Matching Time in OPS5 Production Systems," IEEE Transactions on Software Engineering, Vol. 30, No. 7, pp. 448-457, July 2004.

[76] White paper, "Benchmarking CLIPS/R2", Production Systems Technologies, Inc. http://www.pst.com/benchcr2.htm

[77] White paper, "Benchmarking OPSJ", Production Systems Technologies, Inc. http://www.pst.com/benchopsj.htm

[78] L. Chisvin and R. J. Duckworth, Content-addressable and associative memory: Alternatives to the ubiquitous RAM," IEEE Computer, vol. 22, pp. 51--64, July 1989.

[79] L. Sourdis and D. Pnevmatikatos., "Pre-decoded CAMs for Efficient and High-Speed NIDS Pattern Matching," in *Proc. of FCCM*, 2004.

[80] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, and V. Hogsett, "Granidt: Towards Gigabit Rate Network Intrusion Detection Technology", in *Proc. of FPL*, 2002.

[81] H. Liu, "Efficient Mapping of Range Classifier into Ternary-CAM", Proc. Hot Interconnect 10, Stanford, 2002

[82] Martin Roesch, "Snort- Lightwiegth Intrusion Detection for Networks," *Proceedings of LISA'99: 13th System Administration Conference*, Nov., 1999, pp229-238, Seattle, Washington, USA

[83] Snort Website: www.snort.org

[84] Jacob J. Repanshek, Master's thesis, "A Multi-Gigabit Network Packet Inspection and Analysis Architecture for Intrusion Detection and Prevention Utilizing Pipelining and Content-Addressable Memory", University of Pittsburgh, 2005

[85] Snort version 1.9.0 source code, http://www.Snort.org/dl/

[86] "75K62100_NSE Datasheet Brief, Integrated Device Technologies", June 24, 2003, Available at: http://www1.idt.com/pcms/tempDocs/75K62100_DS_34483.pdf

[87] Martin Roesch, "Snort Users manual version 1.9.x," Apr. 2002, http://www.Snort.org/docs/SnortUsersManual.pdf

[88] Fuller, et al, "Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy," RFC 1519, Sept, 1993, ftp://ftp.rfc-editor.org/in-notes/rfc1519.txt

[89] Martin Roesch, "Snort Presentation," 2001, http://www.blackhat.com/presentations/bh-usa-01/MartyRoesch/bh-usa-01-Marty-Roesch.ppt Access date: Mar. 30, 2003

[90] M. Fisk and G. Varghese, "Fast content-based packet handling for intrusion detection," Technical Report CS2001-0670, University of California, San Diego, Department of Computer Science and Engineering, June 2001. Available also online: http://public.lanl.gov/mfisk/papers/ucsd-tr-cs2001-0670.pdf , Access date: Feb. 7 2003

[91] Jay Fenlason and Richard Stallman, "GNU gprof - The GNU profiler," Available online http://www.gnu.org/manual/gprof-2.9.1/html_mono/gprof.html, 26 pages Access date: Feb. 26, 2003

[92] S. Li, J. Torresen, O. Soraasen. "Exploiting Reconfigurable Hardware for Network Security," Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 2003.

[93] Cho, S. Navab, W.H. Mangione-Smith. "Specialized Hardware for Deep Network Packet Filtering," FPL 2002, LNCS 2438, pp. 452-461, 2002.

[94] M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, V. Hogsett. "Granidt: Towards Gigabit Rate Network Intrusion Detection Technolog," FPL 2002, LNCS 2438, pp. 404-413, 2002.

[95] CLIPS: A tool for building expert system, http://www.ghg.net/clips/CLIPS.html, Access date: Aug.10, 2006