

# **Optimization of IDS systems**

by Lubomir Nistor  
November 2002

## Introduction

Dear reader,  
congratulations for getting your brand new IDS system, but by reading this I suspect that you are not happy with its performance. This paper doesn't tell you how to make your system faster it should serve as an overview of various technologies an IDS can be based upon and their advantages.

For developers this paper serves as an advice on how to design a good engine that is able to perform as good as best of breed commercial software. Market expects IDS systems to perform at higher speeds but still be flexible enough to detect today's or any future intrusions. Such challenge can't be won by other means that modifying the engine to suit the needs of new expectations.

This paper should serve as an overview of many options in different technologies being used to design an IDS engine. Implementation of each has its advantages and weaknesses. By utilizing each technology to its limits an IDS can be more successful than other market players.

This way or the other this paper helps you to understand what it takes to choose or design a good engine. Various technologies have various advantages and knowing them is important not just for designing IDS but also integrating systems and knowing their weaknesses.

## Rule based systems

Many IDS solutions on the market are based on expert systems engine and knowledge in a rulebase. Their performance and flexibility depends on the knowledge representation. The more complex it is the longer it takes to process each packet. In order to achieve better performance the IDS system has to have more resources (CPU, memory, ..); minimize the rulebase or optimize the knowledge representation. The first option costs money and is not always applicable. Second option decreases the detection capabilities of an IDS. So in order to increase system performance developers have to optimize the knowledge representation by minimizing the amount of rules that are applied on one packet.

### **Organization in a list**

This is very common in today's IDS systems as development and implementation is much easier than other methods. As an example SNORT 1.9 is placing its rules in a list.

```
while(rule != NULL)
{
    if(EvalPacket(rule->RuleList, rule->mode, p))
    {
        ...
    }
    rule = rule->next;
}
Main detection routine in snort 1.9
```

In SNORT the list is defined in RuleListNode and distinguishing between 4 known packets (IpList, TcpList, UdpList, IcmpList) with each node defined in RuleTreeNode with extra options in OptTreeNode. Such engine is efficient in case of not many rules in each list, but with more rules being added to the rulebase, it is advisable to split the lists into smaller entities (but not too many entities and this is done in one *switch* statement and may also slow down the analysis)

This means that every packet has to pass through all the rules in one list. If we had 2 rules in the rulebase one for TCP port 139 and one for TCP port 23 we would have to analyze both rules if we get one or the other type of a packet. In case of many rules using content option it may slow down the analysis process.

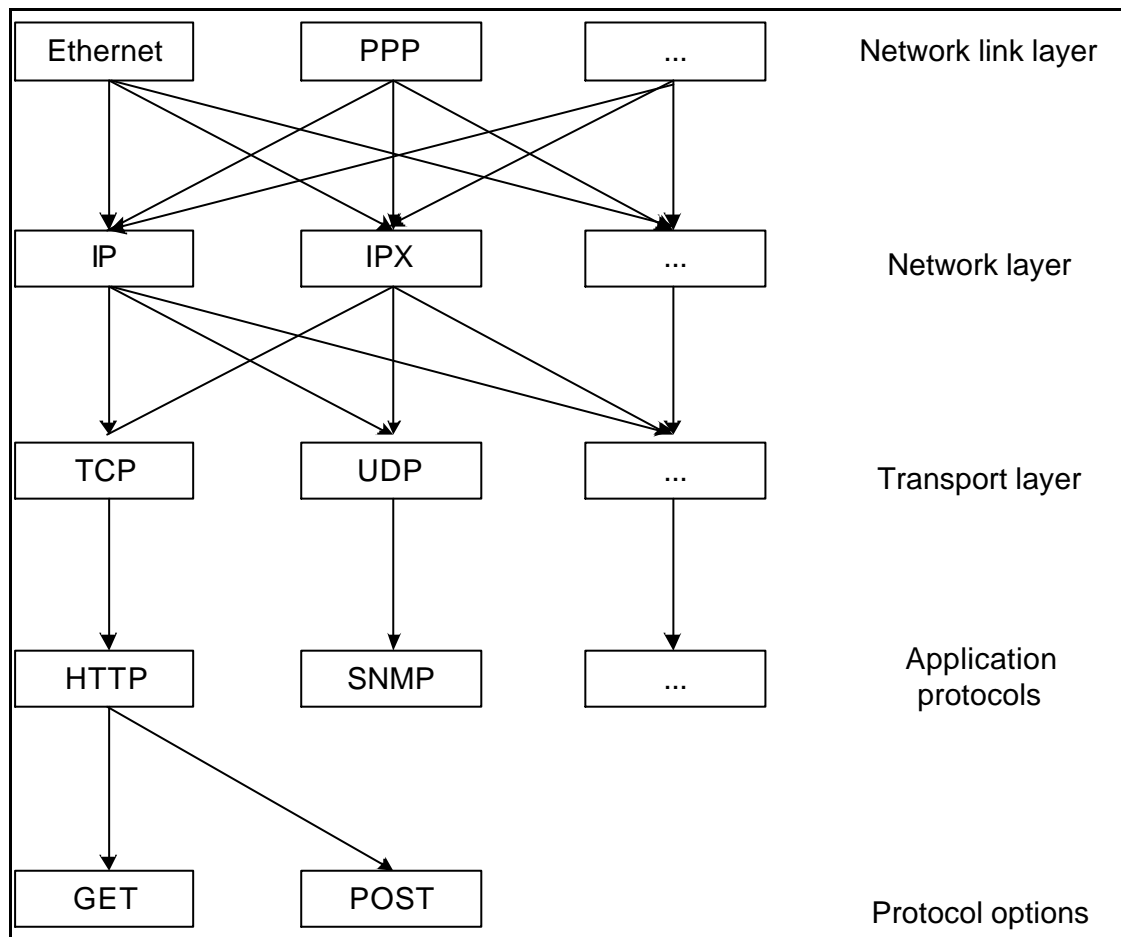
### **Not sorted tree**

A tree has to be sorted in one or the other way in order to move from the list type rulebase representation. If we decide not to preprocess the rulebase in order to optimize it we have to leave it to the administrator. This means that system maintenance would get more complex and packet processing engine too. Speed advantages would depend on the organization of the rulebase by system administrators. The complexity of such design would make systems using this rulebase organization unmarketable as not many administrators design their own signatures and even fewer of them do a proper design and optimization of the rulebase.

## Tree sorted by protocol

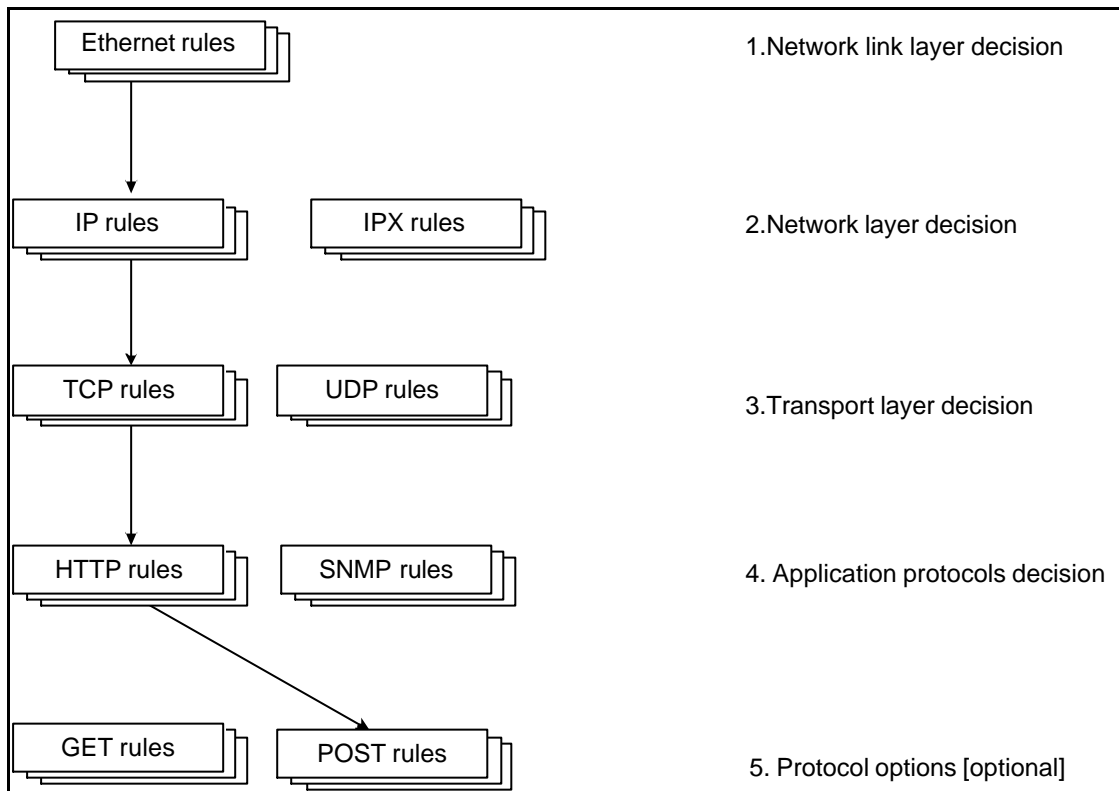
More specific sorting of rule representation tree would offer greater processing speed as the engine can be more specific and static. If we split the list rulebase into 2 parts (one for TCP and one for UDP) we double the processing speed (from the general perspective). Splitting it deeper into specific application protocols offers even higher speeds when the tree is designed properly.

An example of tree organization is shown on the picture below:



The engine for such a tree should look like this:

```
if (packet_ethernet (p))
{
    EvalPacket(tree->ethernet->RuleList, rule->mode, p);
    if (packet_IP (p))
    {
        EvalPacket(tree->ethernet->IP->RuleList, rule->mode, p);
        if (packet_tcp (p))
        {
            EvalPacket(tree->ethernet->IP->TCP->RuleList, rule->mode, p);
            if (packet_HTTP (p))
            { do_http_stuff }
        }
    }
}
```



Description of each step taken in the engine should help to explain the functionality. Identification of the rule position in the tree may be based on the protocol type (second field in the rule).

### Step 1.

Analyzing the link layer protocol may be sometimes useful but is not used very often in rules design (snort plans to implement it). In this step all the packets with specific link layer are inspected by rules only for this layer. So if somebody designs a rule:

```
alert arp ! $LIST_OF_HOME_MACs -> any (msg: "unauthorized MAC detected");
```

All such rules would be inspected in this step.

### Step 2.

Here should be all the rules analyzing protocols on a network layer mainly IP.

```
alert ip $EXTERNAL_NET any -> $HOME_NET 139 (msg: "unauthorized access to netbios ports", content:"|00 0C|");
```

All the rules on this tree leaf should be looking into IP header and in case the protocol type is not supported by deeper layer leafs also the rules

### Step 3.

Analysis of transport layer protocols is not really necessary as they may be included

in step 2. but just as an example it should help.

All the unsupported ports should be also part of this rule list, which allows the administrator to use the engine for anomaly detection (by defining ports that systems don't use as alert conditions)

#### Step 4.

This is the most important decision step where packets are separated by application service. Decoding incoming packets on the application layer should help detect anomalous packet structures (used when scanning hosts or testing services)

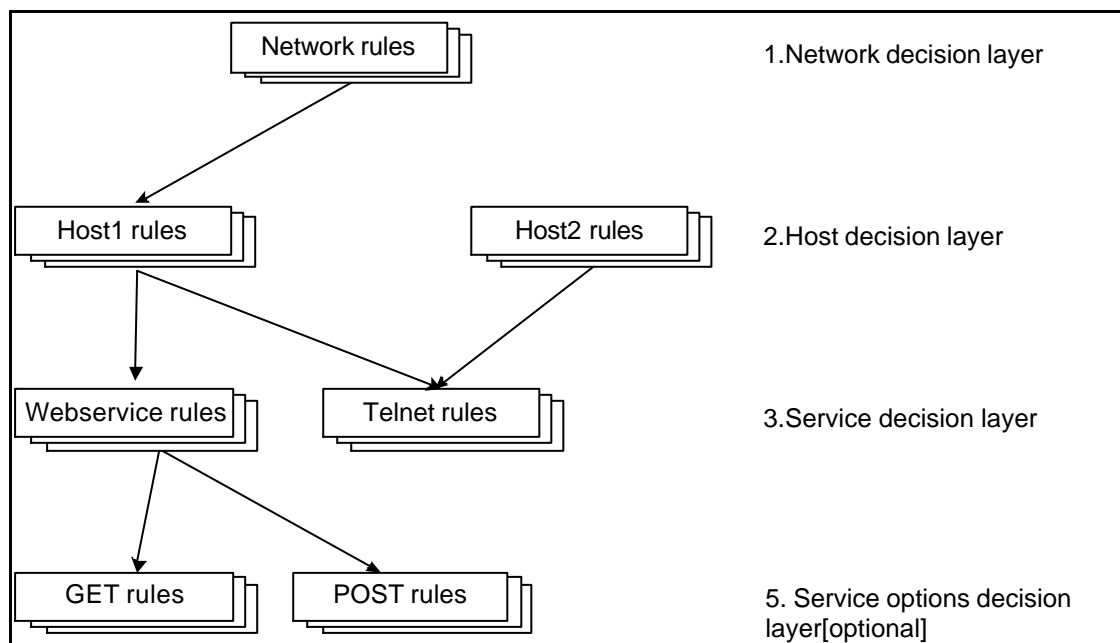
```
Alert http any -> $WEB_SERVER (client: "Internet Explorer 5.0"; cookie ; keepalive; URL: ".php");
```

#### Step 5

This step takes the tree structure a bit more deeper into optimization especially for big throughput systems, which have many rules. Decoding contents of the packets received is not always an advantage as even preprocessing takes a lot of time, but this way some evasion techniques would not work (fx. URL encoding).

### **Tree sorted by network structure**

Small IDS implementations protect a very small network with several hosts. Such solutions require an engine that has a small number of protected IP addresses and concentrates more on services. By distinguishing between networks and hosts the engine can perform better than the previous one. Just as an example an IIS server with many contents inspection rules doesn't have to be in the same part of the rule tree as other hosts with web services. By distinguishing between hosts it is able to make decisions specific to the services that are on the host.



**Perform:**

1. Do if packet is part of network
2. Do if packet has IP address
3. Do if packet has service
4. Do if packet has command or option

Here an advantage is that the rulebase tree may also represent host IDS rules or other important intrusion detection conditions which may be detected by various agents. Such tree sorting has higher efficiency in small networks or when NIDS agents are placed on the actual servers that run critical business applications.

## Anomaly detection based systems

Behavioral or statistical IDS technologies are gaining more and more importance as they are able to detect known or unknown attack by identifying behaviors or anomalies that differ from the standard system behavior or communication.

This is mostly done by analyzing statistical trends and watching certain variables.

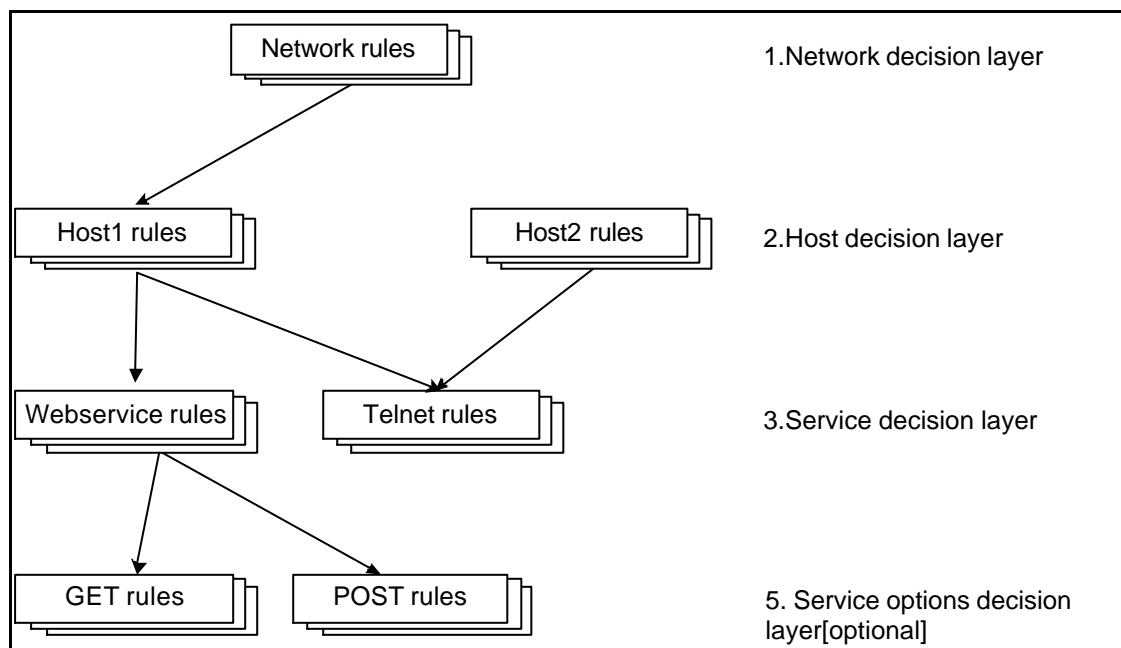
There are also other possibilities to achieve “anomaly” based IDS system. By defining acceptable behavior in the rulebase a solution can be described as “anomaly based” as it produces alert when there is a not acceptable traffic passing through the engine.

### **Rule-base behavior definition**

If you use a typical IDS engine you have to define rules that detect intrusions. That means you specify what you do not expect. But if you write a rulebase by specifying what you expect and produce an alert if not, you may achieve a system that may be called “anomaly detection” based.

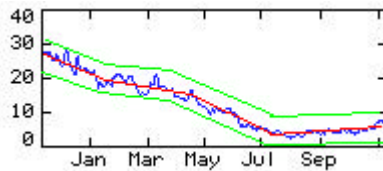
In order to do that you just need to use the engine available if it allows definition of “allow” or “accept” rules.

Another option is to negate the result of detection routine if sources are available.. In both options rule redesign has to happen.



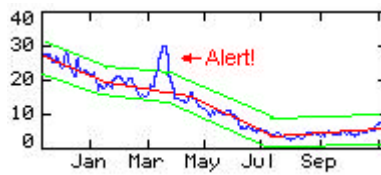
### **Statistical behavior definition**

Another possibility for anomaly detection system is to use statistical modeling. This is not always exact and as good in identifying attacks as a rulebase system, but by making the model of the network more and more precise it is possible to achieve very good results in more static networks where changes don't happen so often.



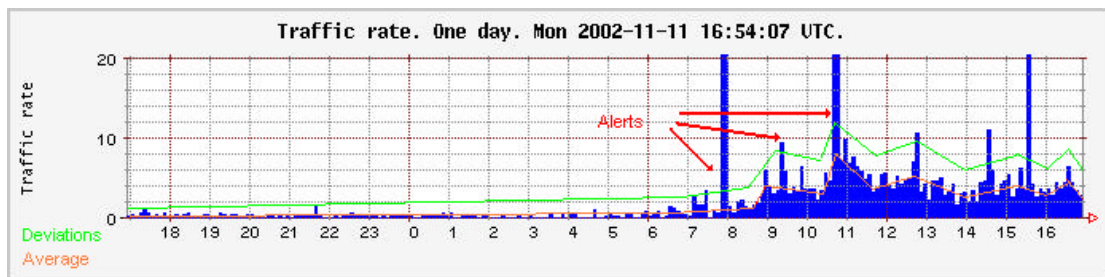
Standard behavior

On the picture above you can see a communication statistics for one protocol during a longer period of time. A statistical system should determine average behavior (red line) and watch for any deviations (acceptable maximum is shown as green lines) that exceed the acceptable maximum.



Alert detection

Here you can see a deviation that was caused by excessive communication with that protocol (fx. A user was transferring company's database to another server).



Another alert detection

Another example for general traffic of one switch port shows that at some points in time a web-server was very communicative, which means that somebody was downloading large files from server or somebody was trying to do DDOS flooding.

A big question by statistical systems is which features to monitor. By defining too many options to watch it could have a very similar problem as rule-based systems, that is analyzing packet by too many functions and slowing down the processing.

A statistical model can be described as:

$$F(p) = \sum ( importance_i * max\_deviation_i * packet\_feature\_test\_f_i(p) )$$

## **AI systems**

Expert systems process the packets in a similar way as network experts do by using know-how stored in the rulebase and engine designed to analyze the packets being the input.

Artificial intelligence is another step forward in the progress but it is not an easy task to do. AI is a similar technology to the tree-like decision making, but it is a bit more dynamic system than expert systems. The rulebase is replaced by learning the AI network by telling it what input is good and what input is bad. Vendors may already sell “initialized” or learned network that needs just a slight adjustment to fit the customer's needs.

Here speed optimization doesn't come into question as the AI processing is much more CPU intensive as simple rule-based packet processing and also such systems require experts in order to implement it in the production environment . In dynamic environments it would be constantly being modified to fit the requirements of the client and therefore I doubt that such systems would be market ready in near future.

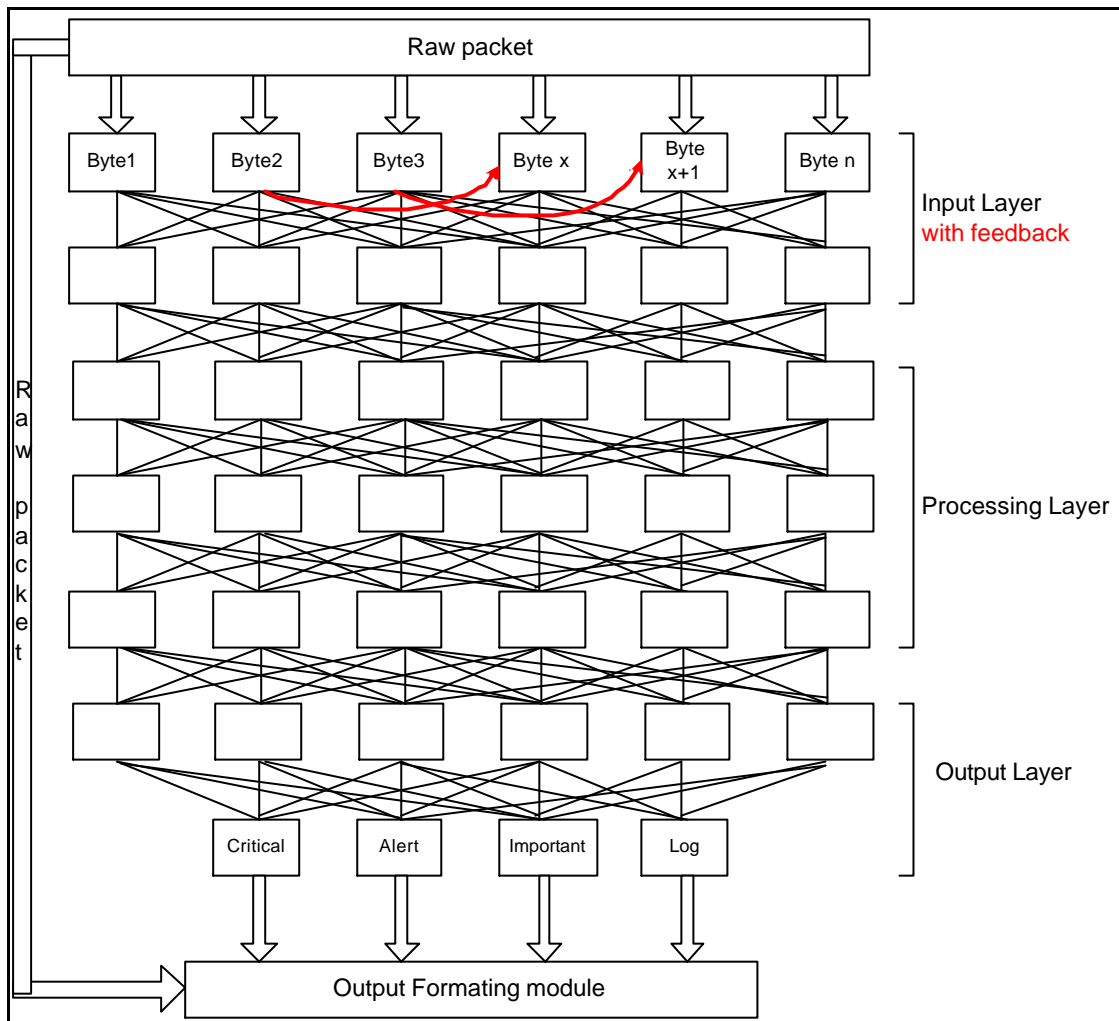
Lets just suppose somebody would try to develop an AI IDS and would try to optimize it. I can think of 2 engine types that may come into question:

### ***General AI engine***

This engine type takes the raw packet and analyzes each byte of it. In order to help the input layer identify the protocol and packet structure there should be feedback introduced that would affect the value of neurons in input layer.

Picture: AI engine with raw entry (entry layer for each byte with feedback from packet type fields to payload entry layer)

In case of new protocols being introduced in the network the AI network doesn't have to be modified (although in some cases there have to be extra feedback relations added).

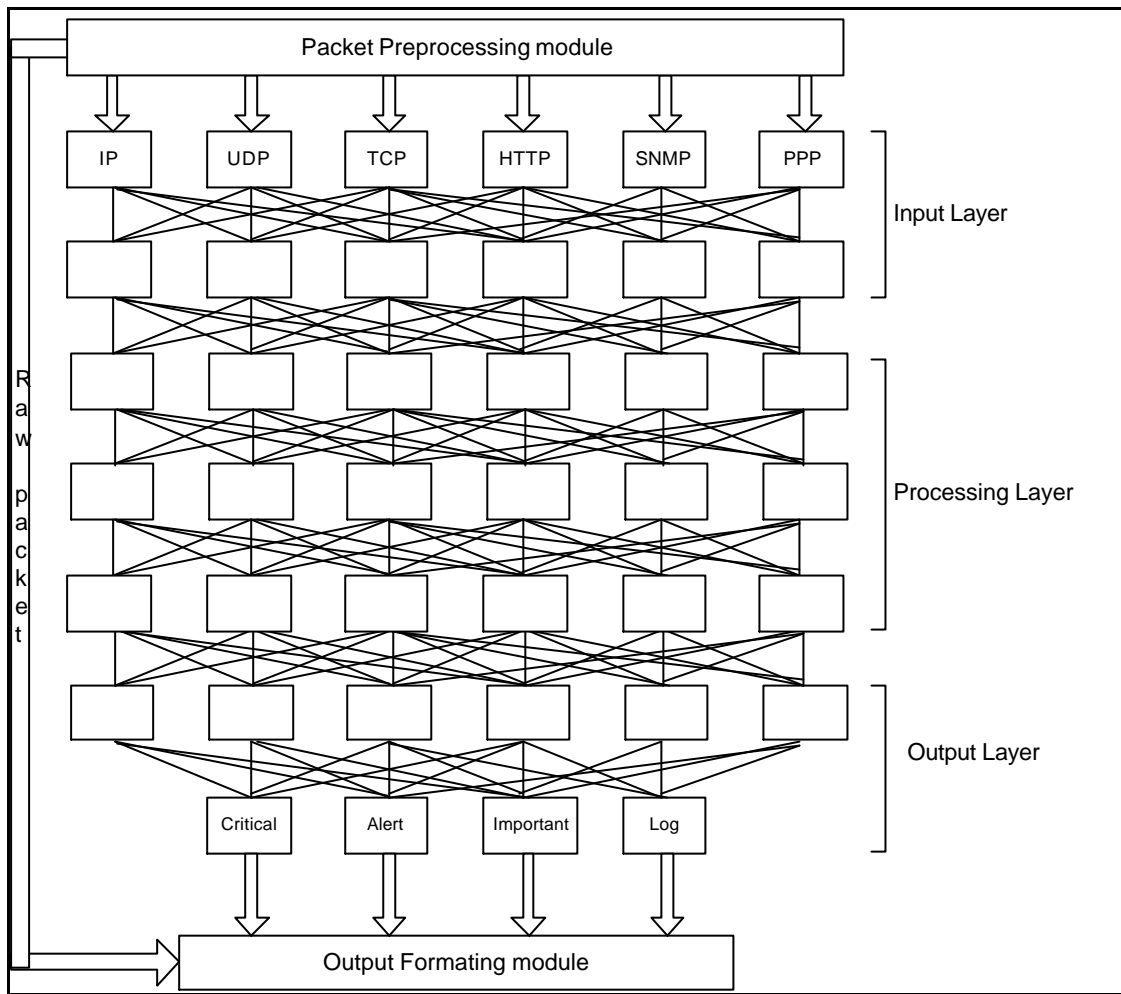


## ***Preprocessed AI engine***

Processing of a raw packet requires less time when done by external process that puts the preprocessed data into the appropriate input layer parts. Each type of protocol has to be present in the input layer and every option and command would be present as an input neuron.

Picture: AI engine with packet preprocessed (entry layer for every packet type)

Another alternative here may be preprocessing and identifying anomalous states in each protocol and placing only these values to the input layer.



## Conclusion

Many years ago companies in the field of security raced for the best antivirus engine. This race was very hard but after several years all the companies reached almost the same position in the research and their engines have many things in common (it is also thanks to reverse engineering and buyouts).

Similar progress should be expected in IDS industry, although during today's recession people with new ideas are not able to start their companies in order to design their own systems and show the market another solution.

By using open-source solutions market may be able to adjust itself if skilled engineers have sufficient time to do modifications and optimization of open-source engines (and I doubt they have time for that).

This paper should give some extra ideas to those that are lucky and have a job where they work with IDS systems and are able to test them out in real life.

I hope with this paper I can accelerate communication and design in order to bring the market to a mature state and allow security architects to be able to implement very effective IDS solutions.