

# Deep Packet Inspection Using Message Passing Networks

Divya Jain, Vasanta Lakshmi K, and Priti Shankar

Indian Institute of Science

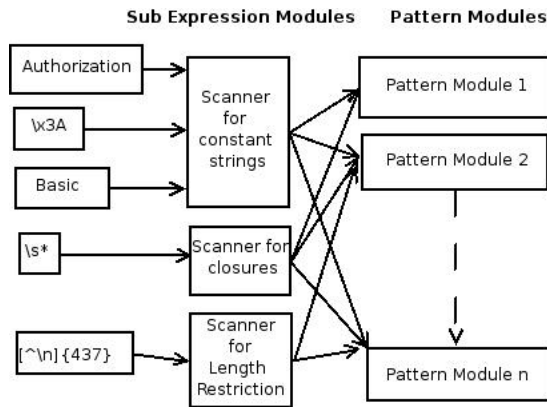
**Abstract.** We propose a solution based on message passing bipartite networks, for deep packet inspection, which addresses both speed and memory issues, which are limiting factors in current solutions. We report on a preliminary implementation and propose a parallel architecture.

## 1 The Problem, Our Solution and Results

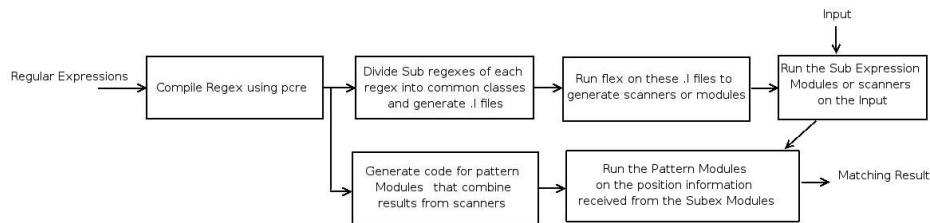
Packet content scanning at high speed is crucial to network security and network monitoring applications. In these applications, the packet payload is matched against a given set of patterns specified as regular expressions to identify specific classes of applications, viruses, protocol definitions, etc. Unfortunately, the speed requirement cannot be met in many existing NFA based payload scanning implementations because of the inefficiency in regular expression matching. Deterministic finite automata (DFAs) for certain regular expression types suffer from state blow up limiting their practical implementation. To solve the problem of state blow up we propose the following two part process.

1. In the first step the regular expressions are divided into subexpressions and these subexpressions of all regular expressions are then categorized into “sub expression modules” depending on their type. These modules are essentially scanners that run on the input and return the positions where the subparts occur in the input. Every regular expression is composed of one of constant strings, closures, length restrictions, a class of characters, a query or a combination of these. Thus the regular expression can be broken down into these components and all components of same type can be combined into a single module. For example all constant strings occurring in all regular expressions will be in one module with a single DFA scanning for constant strings in the input. Many of these components especially closures and length restriction are common to several regular expressions and thus running a single DFA for all of them is beneficial. We have a single module for every type of subexpression. The current number of subexpression modules for this implementation are 6.

2. In the second step we construct *pattern modules* for each regular expression, which collect events consisting of subexpression matches along with matching positions (each such pair termed as an event) generated by the sub expression modules in the first step and string them together to find a match for the actual regular expression. Each such pattern module is modeled as a *timed automaton*.



**Fig. 1.** Fragmenting RE 'Authorization\s\*\x3A\s\*Basic\s\*[\n]{437}'



**Fig. 2.** Block Diagram of our solution

Each subexpression knows which pattern messages it needs to send messages to so each timed automaton gets only events of potential interest to it.

In a timed automaton each symbol represents a timed event, namely a sub expression of the regular expression along with its “time” or position in the input stream, transitions being based on such events. The pattern modules are independent of each other and hence can be run in parallel in a hardware setup or on a parallel machine. Currently there is one pattern module per regular expression. As the number of regular expressions is large, in future, we plan to combine multiple regular expressions into single pattern modules based on the subexpressions they share. As a trial setup we have run the implementation for a selected set of 32 regular expressions from more than four thousand such rules that make up SNORT PCRE rulesets. The flex generated scanners are in C and the rest has been implemented in C++. The expressions chosen cover almost all types of regular expressions. The number of states for the traditional DFA for each of the 32 regular expressions ranges from 30 to almost  $2^{400}$  where 400 is the length restriction in one of the expressions. The total number of NFA states for these regular expressions is 11924. In our solution, the total number of states for all DFAs in the sub expression modules is just equal to 533. The number of states in the pattern modules is equal to total sub expressions which is 300. This solution appears scalable though proper parallel communication protocols and a fast simulation of a timed automaton are critical for efficient functioning. We plan to do the simulation of the parallel implementation on the Blue Gene-L machine and later will examine hardware implementations.