# Traffic-Aware Design of a High-Speed FPGA using Network Intrusion Detection System

**Ravi D. Bhadja[1], Prof. N. M. Wagdarikar[2]**

*Pursuing M.E (VLSI and ES), SKNCOE, Pune, Maharashtra, India. [2]Asst. Prof. ECE department, SKNCOE, Pune, Maharashtra, India.*

***Abstract : Network Intrusion Detection Systems (NIDS) are critical network security tools that help protect distributed computer installation from malicious users software-based NIDS architectures are becoming strained as network data rates increase and attacks intensify in volume and complexity. Now a days, researchers have proposed using FPGAs to perform the computationally-intensive components of a NIDS. Here, we present the next logical step in NIDS architecture: the integration of network interface hardware and packet analysis hardware into a single FPGA chip. This integration permits better customization of the NIDS as well as a more flexible foundation for network security operation. To demonstrate the benefits of this technique, we have implemented a complete NIDS in a Xilinx Vertex II/Pro FPGA that performs in-line packet filtering on multiple Gigabit Ethernet links using rules from the Snort attack database.***

***Keywords – FPGA, Network Intrusion Detection System, in-line packet filtering.***

## 1. INTRODUCTION

Network Intrusion Detection System is a device or software application that monitors network or system activities for malicious activities or policy violation and produces report to a management. Software-based NIDS, such as the widely employed software implementation of the Snort NIDS, cannot sustain the multi-Gbits/sec traffic rates typical of network backbones and, thus, are confined to be used in relatively small-scale (edge) networks. For high-speed network links, hardware-based NIDS solutions appear to be a more suitable choice, but the hardware implementation needs to permit the latest update of the supported rule set, so as to cope with the continuous emergence of new different types of network intrusion threats and attacks. Field-programmable gate arrays (FPGAs) are, thus, appealing candidates. Though, an FPGA-based NIDS can be easily and dynamically reprogrammed when the content matching rules change.The specific contributions of this work can be summarized as follows:

1] Snort rules analysis and relevant classification policies: We analyze the whole rule set of Snort, to organize such set into disjoint subsets, identified by suitable combinations of packet header fields. For instance, the rule subset in charge of detecting possible exploits against http servers (protocol = TCP, destination port = 80) obviously differs from the set of rules to be employed by another protocol such as FTP or SMTP; but, perhaps less obviously, also differs from the subset dedicated to analyze threats still for the http protocol, but against web clients (protocol = TCP, source port = 80). Such an analysis yields the classification policies exploited in the dispatching of traffic toward the hardware modules, each supporting one or more sub-sets.2] Also, in the software implementation of Snort , rules are grouped with the help of port/protocol, and for each perticular packet, only the group of rules corresponding to the port/protocol of the packet are checked. This rule partitioning will reduce memory consumption and CPU usage of Snort rule-set. Unlike the software implementation, in our case this partitioned is the first step for applying traffic awareness.

2] Real world traffic analysis for HW module sizing: We offline analyze real-world traffic, provided by an Internet Service Provider, to quantitatively assess how traffic splits according to the envisioned classification policies, determine the expected worst-case per-class throughput and, thus, set forth the relevant input rate requirements for dimensioning each content matching HW module. In essence, we apply, for an HW-based development, a methodology similar to the one proposed in [3], where an adaptive algorithm depending on the traffic mix is used to optimize a software based IDS. We stress that the goal of such an analysis is not to provide a once-for-all system dimensioning, but, rather, to suggest a methodological approach. Indeed, variations in the traffic mix do occur during the operating lifetime of the NIDS and may also depend on the specific operator's deployment. This does not appear to be a practical concern, as in any case the synthesis of the content matching engine must be rerun at every rule set update (order of once per week), whereas

variations in the traffic mix are shown much slower, in the order of many weeks. And when significant variations in the traffic mix are detected, the resulting system redesign can be conveniently accounted for, while performing the synthesis associated to the periodic rule update.

3] HW module implementation and relevant tradeoffs: We perform several constrained syntheses (with respect to speed and area) of the different SMEs, to gather insights in the emerging area/speed tradeoffs for the specific NIDS rule set synthesis. If multiple copies of the same SME are used to achieve a higher throughput, the choice between area or speed optimization of the engine is not unique, but strictly depends on the circuits to be implemented, and in some cases, the emerging area-delay tradeoffs are little unexpected . Use of multiple copies of low-speed SMEs allows to make an optimization between the area of the single engine, its maximum operating frequency, and the overall throughput.

## 2. SYSTEM MODEL

As anticipated in the introduction, our proposed system comprises multiple string matching modules. These modules are further organized into clusters, of suitably sized so as to handle the expected per-cluster traffic load. Packets balancing will be done across clusters on the basis of policies implemented in a block called dispatcher.

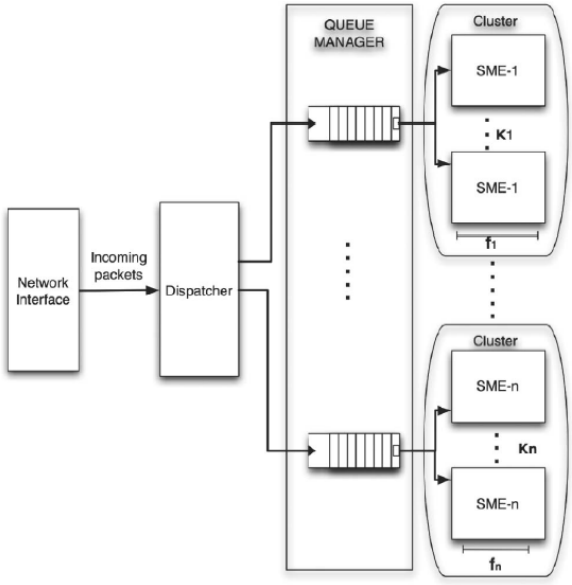


Fig.1 implementation of string matching system [1]

The main blocks of the system are:

- **Network interface:** It collects packets from the network link under monitoring;
- **Dispatcher:** It provides a header-based packet classification, whose result is used to determine to which specific string matching cluster the packet is transmitted;
- **SMEs**: blocks performing string matching; their design is identical (as described in the previous section), but the content searching rules synthesized in SMEs belonging to different clusters differ and specifically depend on the type of traffic routed to the considered cluster. A generic string matching system is composed of n cluster, each one clocked at a specific frequency fi and composed of Ki identical SMEs.
- **Queue manager:** This block provides a queue for each SME cluster. The queue provides the buffering of packets to cope with packet bursts. The queues can be realized by using external memories to provide enough space. The memory can be partitioned as a set of circular buffers, each one controlled by two pointers. A control FSM, realizing a round-robin policy, allows using the memory as a set of independent queues. Since the SME cluster may be clocked with a different frequency, with respect to each other and to the queue manager, asynchronous FIFOs for clock decoupling are deployed between the queue and the SMEs.

Since, multibyte SMEs do complicate the internal design, the queue output uses 8 bits. Conversely, the interfaces between the remaining modules can be implemented using multiple characters at a time.

The architecture shown in Fig. 1 is very flexible and general. The resulting operation in fact depends on a configuration setting that includes the following decisions and parameters:

- Dispatcher classification policy;
- string matching rules loaded over each cluster of engines;
- operating frequency of each cluster; and
  number of SMEs deployed in every cluster.

## 3. PREVIOUS WORK

Software-based NIDS, such as the widely employed software implementation of the Snort NIDS [1], cannot sustain the

multi-Gbits/sec traffic rates typical of network backbones and, thus, are confined to be used in relatively small-scale (edge) networks. For high-speed network links, hardware-based NIDS solutions appear to be a more realistic choice, but the hardware implementation needs to permit the frequent update of the supported rule set, so as to cope with the continuous emergence of new different types of network intrusion threats and attacks.

## 4. PROPOSED METHODOLOGY

we are interested in exploring the performance gains that may be achieved by dispatching different traffic types to different clusters, consistently distributing different content matching rules over different engines, independently optimize the area-frequency tradeoff for each deployed engine, and dimensioning each engine depending on the traffic load conditions. For simplicity, we take a practical three-steps design, organized as follows:

Step 1: Rule set distribution and relevant packet dispatching policy. The first step, is to distribute different content matching rules across multiple engines. Such distribution is driven by two practical requirements:

1) Permit an elementary dispatching policy, based on simple protocol header information, meanwhile 2) attempt to obtain (as much as possible disjoint) subsets of size smaller than the whole rule set. Our proposed classification, indeed relies on trivial protocol/ port information, thus permitting a straightforward implementation of the dispatcher. It is worth noting that per-protocol grouping of string matching rules is the most natural direction, as in practical NIDS such as Snort, rules defined for the same protocol not rarely share common substrings (for instance, the string “HTTP” requires to be matched by most rules applied to protocol: TCP and destination port: 80) and, hence, may yield savings in the subsequent HW circuit design.

Step 2: Per-engine optimized HW design. For each specific engine (and its subset of different rules), Quite surprisingly, such tradeoff significantly depends on the specific rule set considered. The output of this second stage design is the frequency at which each engine is implemented.

Step 3: Traffic-load-based system dimensioning. we perform an experimental analysis of real world traffic devised to provide information about the per cluster load and, consequently, determine how many copies of each synthesized engine are needed to sustain the resulting load.

Obviously, the outlined approach is open to improvements, by using information here exploited for individual steps in a more holistic design procedure (e.g., use traffic information for determining how to distribute rules across engines), although it does not appear simple to move from heuristics to a more formal design methodology. Finally, even if here we refer always to the SMEs described in the previous section, we outline that this method can be generically applied to many of the string matching systems proposed in the literature. For example, also, work in [4] could benefit of a partitioned traffic-aware implementation, since the implementation techniques used to improve performances (i.e., pipelining, parallelism, and memory replication) suffer from the same scalability issues already mentioned in the previous section for other techniques [2], [3]. Instead, packet-level parallelization should be able to better exploit area/delay tradeoff than the classical parallel/ pipelined implementation, and the traffic-awareness could be easily reduce the memory replication.

## 5. CONCLUSION

Network intrusion detection systems (NIDS) are a necessary tool for monitoring and protecting computer networks from malicious users. As network data rates and malicious user sophistication have increased over the years, it is necessary to consider new NIDS architectures that will be able to meet stringent constraints. We can implement a complete and functional NIDS in a commercial FPGA chip. This system can perform in-line packet filtering on multiple Gigabit Ethernet links using intrusion detection rules based on the Snort rule set.

## REFERENCES

[1] S. Pontarelli, C. Greco, E. Nobile, S. Teofili, and G. Bianchi, “Exploiting Dynamic Reconfiguration for FPGA Based Network Intrusion Detection Systems,” Proc. IEEE Int’l Conf. Field Programmable Logic and Applications (FPL), pp. 10-14, 2010.

[2] N. Yamagaki, R. Sidhu, and S. Kamiya, “High-Speed Regular Expression Matching Engine Using Multi-Character NFA,” Proc. Int’l Conf. Field Programmable Logic and Applications, pp. 131-136,2008.

[1] I. Sourdis, D.N. Pnevmatikatos, and S. Vassiliadis, “Scalable Multigigabit Pattern Matching for Packet Inspection,” IEEE

Trans. Very Large Scale Integration Systems, vol. 16, no. 2, pp. 156-166, Feb. 2008.

[2] I. Sourdis, D. Pnevmatikatos, S. Wong, and S. Vassiliadis, "A Reconfigurable Perfect-Hashing Scheme for Packet Inspection" Proc. 15th Int'l Conf. Field Programmable Logic Application, pp. 644-647, 2005.

## AUTHOR'S PROFILE

**Ravi Bhadja** has received Bachelor of Engineering degree in Electronics & Telecommunication Engineering from JNU College, Jodhpur in the year 2012. At present I am pursuing M.Tech. with the specialization of E&TC in SKNCOE, pune, maharastra. His area of interest VLSI and Embedded.

**Asst. Prof. N.M. Wagdarikar,** currently working as assistant professor in Smt. Kashibai Nawale College of Engineering, Pune, Maharastra.