An FPGA Based Implementation Of A Packet Filter

Timothy Whelan

November 10, 2010

Abstract

This paper describes the design of a TCP/IP packet filter implemented on a Spartan-3AN development board. The design uses bit mask matching and binary number range matching techniques to determine the matching of TCP/IP field values to those specified in a user provided rule set. The initial design can accommodate ten rules against which packets can be matched but it is shown that the rule set can be expanded up to around 100 rules to match against packets. Currently the design is suitable for operating on TCP/IP packets but the design of the filter's module allow for the support of other transport protocols to be easily facilitated. The filter design in this paper is expected to operate successfully at line speeds of 100Mbps or less.

1 Introduction

Data transmitted over networks require much extra information describing how the data is meant to be transmitted. This metadata can provide information useful for determining the origin of the data and what kind of data is being transferred. This in turn allows one to monitor network traffic received and examine the metadata for items of interest. However the task of identifying relevant traffic is a challenge as the process that identifies interesting traffic must keep the time cost incurred when analysing traffic to a minimum to prevent traffic becoming delayed for too long or being dropped completely.

To this end research has been conducted in the field of packet classification to attempt to maximise throughput of such devices or processes and of particular interest in this paper is the use of hardware, notably field programmable gate arrays (FPGAs), to filter network traffic. This paper explores the use of such devices for packet classification and a few of the related algorithms and then describes a partial implementation of an IP packet filter on a Spartan-3AN FGPA development board along with simulations of the modules that form part of the filter.

2 Packet structure

The protocols developed to carry data over networks in various ways each use different pieces of data to facilitate their functions. For example the Internet Protocol (IP) is responsible for transferring data over a network, among other responsibilities, from one node to another [5]. To accomplish this the IP protocol uses IP addresses to address the source and destination network nodes that the data must be transmitted between. Since different protocols perform different functions multiple protocols are required with the data to be transmitted encapsulated in one protocol and the resulting data is encapsulated in successive protocols until enough information is present within the resulting encapsulated data to transmit the data over the network (for the case of IP the encapsulated form is called an IP packet) [5].

A typical example of this, which is the particular focus of the filter described in this paper, is the TCP/IP stack – the encapsulation of data within a TCP datagram encapsulated within an IP packet. For the specific application of the filter in this paper IP packets travelled over Ethernet and were encapsulated within Ethernet frames. Diagrams illustrating the structure of Ethernet frames, IP packets and TCP datagrams are shown in figures 1, 2 and 3 respectively.

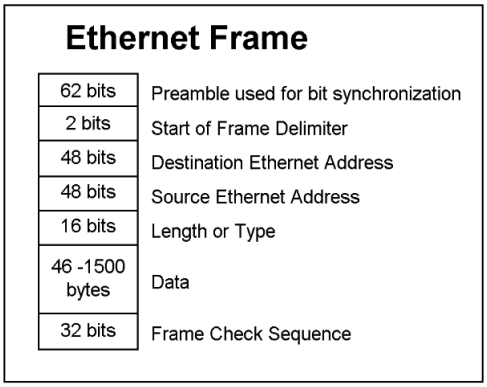

Figure 1: The Ethernet frame structure taken from [1].

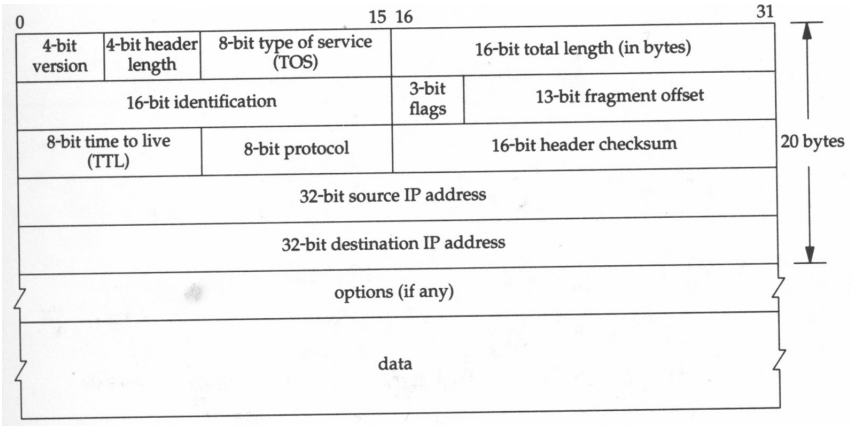

Figure 2: The IP Packet structure taken from [5].

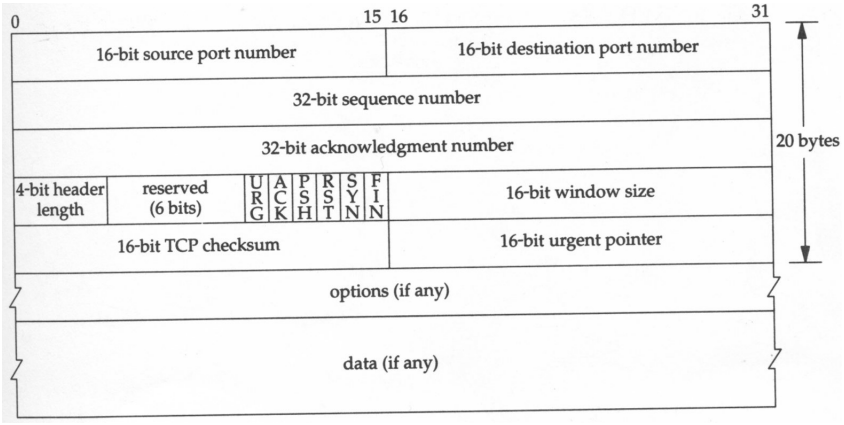

Figure 3: The TCP datagram structure taken from [5].

3 Problem statement

The problem of packet classification is a relatively simple one. N-fields of a packet are chosen as relevant and together they form an N-tuple and for each packet that passes through the filter the N-tuple of the packet is matched against N-tuples with specific values that are stored in a list of such N-tuples (termed a rule set). The core issue of packet classification is the efficiency at which packets can be matched which we seek to maximise for the reasons mentioned in the introduction. Once it has been determined that a packet does or does not match a rule a course of action for that packet can be decided.

4 Packet classification techniques

Various techniques have been developed to try and accelerate the classification process. Briefly, the two techniques discussed here are the use of grid-of-tries, based upon tree data structures, and the use of bit vectors, which takes a geometric approach.

4.1 Grid-of-tries

This approach is described more formally in [4] but a basic description is provided below. Suppose an example rule set is as in table 1 (note that this initial set represents only source and destination addresses).

Table 1: Example rule set (from Srinivasan et al)

Filter	Destination	Source
F1	0*	10*
F2	0*	01*
F3	0*	1*
F4	00*	1*
F5	00*	11*
F6	10*	1*
F7	*	00*

Firstly the destination addresses of the rules are used to construct a tree structure as in figure 4. This enables any incoming packet to be tracked to a node in this tree (termed the dest-trie) except for the open circle which represents an invalid destination address prefix. The leaf nodes of the dest-trie contain pointers to similar trees that represent decision trees for source addresses (termed source-tries).

The source-tries are dependant upon the destination address represented at the leaf nodes of the dest-trie as shown in figure 5. Here the reader can see the pointers from nodes in the dest-trie to relevant source-trie trees. Note that source address trees are repeated when one dest-trie filter is a prefix of another dest-trie filter.

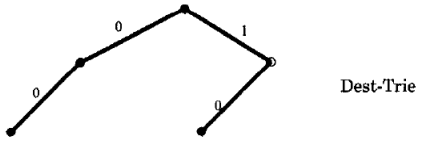

Figure 4: Dest-trie representing the source addresses in the filter set taken from [4].

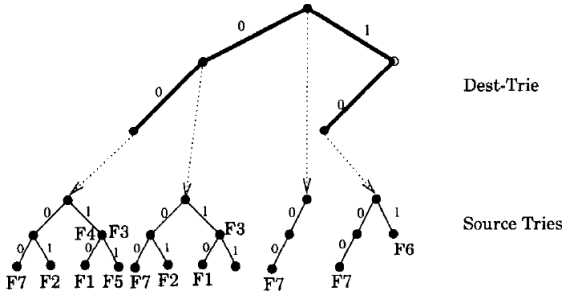

Figure 5: The general trie structure showing the relation between the dest-trie and the source-trie [4].

This grid-of-tries structure can be optimised in many ways and Srinivasan et al go on to describe some of these optimisations. Notably, the memory requirements of this structure can be reduced by not repeating trees in multiple source-tries and search costs can be reduced by placing pointers from one source-trie node to nodes in other source-tries so that when a search down one source-trie branch fails a pointer can be followed to another source-trie and the search can continue without having to perform redundant searches along higher branches before progressing beyond what was already processed at the point of failure [4].

Tries however only work well for 2-dimensional rules [3, 4, 6]. The structure of tries also means that they are suitable for IP address masks but are not suitable for arbitrary port ranges without changing the ranges specifications into prefixes, a task which can significantly increase the number of tries required to adequately describe the desired rules [3, 4, 6].

4.2 Bit Vectors

The bit vector classification technique views the packet classification problem as a geometric problem with each D-field rule specifying a region in D-dimensional space and the classification equates to locating which named region a packet can be said to be placed in [3, 6], an illustration of this view is given in figure 6.

Filtering on two fields (port number and an address) results in a two dimensional filter. The shaded regions with letters represent rules and rules may overlap (darker regions). A brief, simplified, algorithm is adapted from [3] and [6] and described below. Note that there are N filters and D types of fields in each filter.

1. Prioritise the filter rules (if priority is irrelevant then priorities can be arbitrary)
2. For each axis
   (a) Divide the axis into regions bounded by the points where boundaries of regions in D-space that are orthogonal to the current axis may intersect the current axis
   (b) For each region along the current axis
      i. Assign an N-bit bit vector equal to zero
      ii. If region x in D-space, described by rule x, lies on the current region then set bit x of the current region's bit vector to 1

A graphical representation of a geometric view of the classification problem can be seen in figure 6 and a graphical representation of the result of the algorithm can be seen in figure 7.

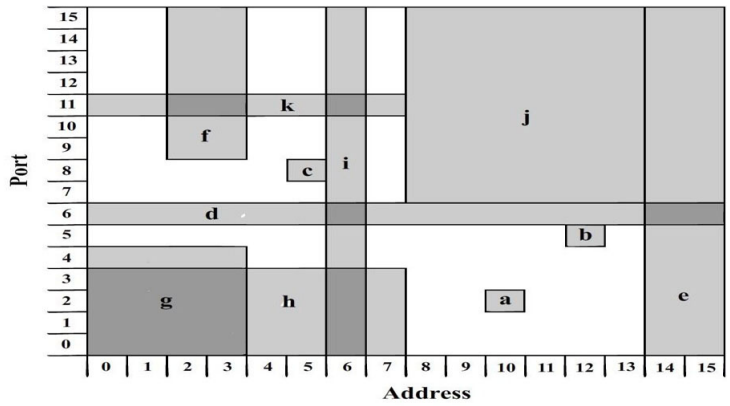

Figure 6: The geometric description of filter rules taken from [3].

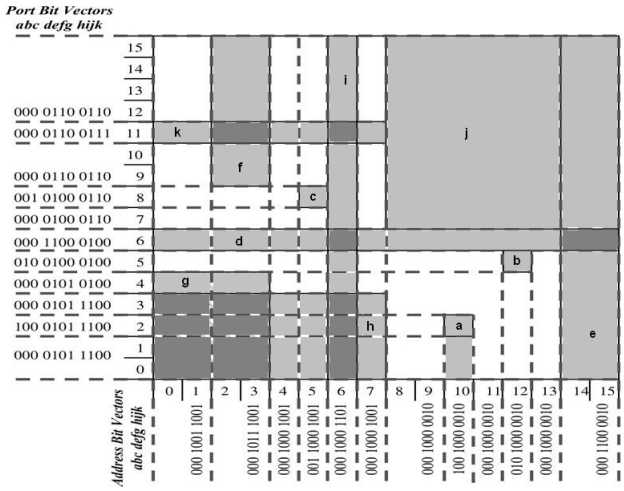

Figure 7: Regions in D-space showing filters and bit vectors for regions along each dimension (N = 11; D = 2) adapted from [3].

When an incoming packet is received its fields are decomposed and each one is processed independently of the others to determine which rules are matched by the packet. Alternatively one can describe the process as locating which region along each of the D axes the packet might belong to.

The D bit vectors (a bit vector result from each field or dimension) are then AND'ed together to give a final bit vector result with each 1 bit representing a rule that is matched by the packet's fields as a group e.g. a packet with port number 8 and address 5 will return bit vectors 001 0100 0110 and 001 1000 1001 as seen in figure 7. When these bit vectors are AND'ed the result will be 001 0000 0000 indicating that rule c classifies the packet.

If the rules in each dimension are ordered according to some priority scheme then the significance of the 1 bits in the final bit vector will match the priority of the rules in the rule set. Li et al [2] describe a bit vector implementation that uses trie structures instead of taking the multidimensional geometric view described above.

5 Design and Implementation

This chapter describes an implementation of a simple packet filter that uses bit masking and large primitive gate structures to perform the rule matching operations. The packet filter described in this work was to be implemented on a Spartan-3AN development kit.

This development board was chosen because it had a variety of interfaces including an Ethernet 10/100 PHY chip on the board and standard RJ45 Ethernet connector, obviously vital components for a project the required receiving data from an Ethernet connection. The board also came with 512MB of DDR2 SDRAM memory which could be incorporated into the design and the multiple interfaces ensured that the board's application could remain versatile.

5.1 Design Overview

The overall design of the packet filter is modularised to allow easy expansion of the design to include more components for expansion of functionality. Modularisation of the design also allowed the developer to leverage the modular structure of packets with each module of the design performing a specific task. The modules in the design are as follows:

- An Ethernet module that reads in nibbles of data from an Ethernet connection and outputs IP packets.
- An IPrx module that reads in bytes of IP packets and records the protocol and IP address fields.
- A TCPrx module that reads bytes of a TCP datagram and records the port number fields.
- A Trie module that matches protocol fields and IP addresses to rules stored in the design.
- A PortBitVec module that matches port numbers to rules stored in the design.
- An Aggregator module that accepts the outputs from the IPrx and TCPrx modules.
- A Count module that records the number of times that packets match rules stored.
- A Report module that reads the counts of rules and reports them over a serial RS-232 interface.

A diagram of the design is shown in figure 8 depicting the relative logical positions of the modules and the signals between them (single bit signals are shown with thin lines and multi-bit buses are depicted with thick lines). The

filter described in this paper has provisions to keep a count of ten rules. It was decided that ten rules was an adequate number of rules to demonstrate the functionality of the filter design.

5.2 Module Descriptions

This sub-section provides a description of how each module in the design operates and how the modules interface with each other.

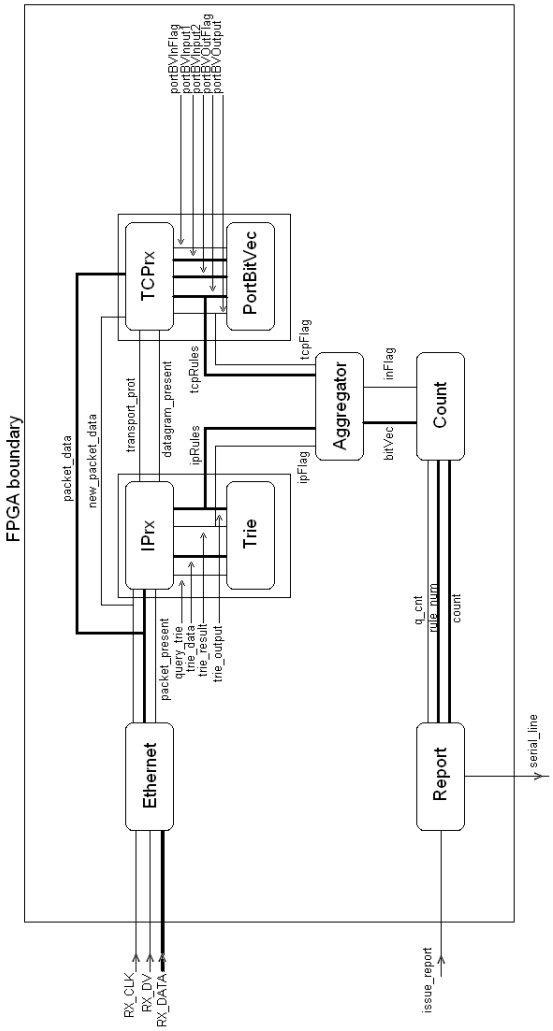

Figure 8: An overview of the filter design showing the modules and the signals between modules.

5.2.1 Ethernet Module

The Ethernet module is responsible for reading data off the wire that arrives in four bit nybbles, with the lower nybble arriving first, and parsing out the MAC addresses and checking the field specifying the network protocol carried in the Ethernet frame. If the network protocol is that of IP a flag (packet_present) is raised and the data nybbles are collected into bytes. Another clock signal (new_packet_data) is raised when each valid byte is received.

A simulation of this process is shown in figure 9.

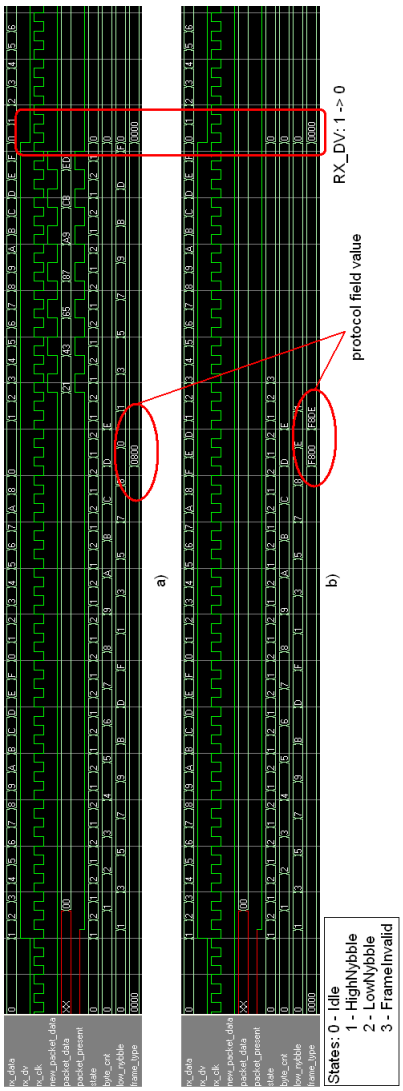

Figure 9: A simulation of the Ethernet module's functionality.

5.2.2 IPrx Module

The IPrx module is enabled by the assertion of the packet_present signal from the Ethernet module and receives the bytes of data passed on by the Ethernet module. The IPrx module parses the bytes of data and records the value of the protocol field and the IP addresses. These field values are then passed to the Trie module which checks the values against bit masks built from the rule set created by the user of the filter. If the bit masks for each field all match for a rule, rule n, then the associated bit in a bit vector is asserted otherwise it is de-asserted. The checks for each field for each rule are all performed in parallel.

The result of the Trie module's analysis is placed on the trie_output bus and indicated by the assertion of the trie_result signal. Once all the IP fields have passed through the IPrx module a packet_present signal is asserted if the IP packet's protocol field indicates that a TCP datagram is encapsulated within the IP packet.

A simulation of this process can be seen in figure 10.

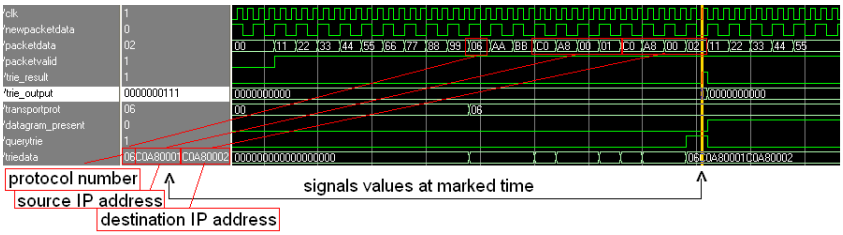

Figure 10: A simulation of the IPrx module's functionality.

5.2.3 TCPrx Module

The TCPrx module operates in a similar way to the IPrx module but is activated by the datagram_present signal from the IPrx module. The TCPrx module records the port numbers contained in the TCP datagram and passes them to the PortBitVec module. For each port number specified in the rule set the PortBitVec module contains a description of AND, OR and NOT operations that can determine if a number (the port number in the TCP datagram) is equal to, greater than or less than a given number (specified in the rule set) in binary form.

However this method uses hundreds of Boolean operations even for a small rule set of only ten rules and is expensive in terms of the number of resources required. The bits of the port numbers are placed at the inputs to these operations and the resulting TRUE or FALSE results indicating a match to a rule are recorded in a bit vector similar to the one returned by the IPrx module.

Figure 11 shows a simulation of the function of the TCPrx module. An interesting factor to note at this point is that the TCPrx uses a byte counter to recognise the bytes of the TCP datagram that contain the port numbers and

as such similar modules can be constructed , based upon the structure of the TCPrx module, to parse datagrams for other transport protocols.

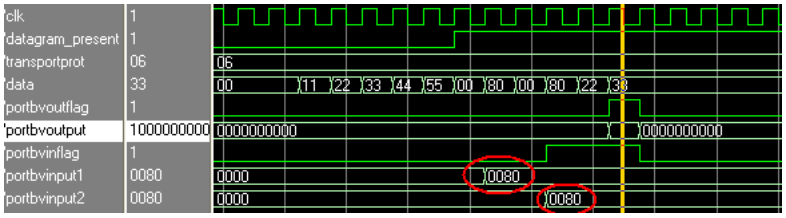

Figure 11: A simulation of the TCPrx module's functionality.

5.2.4 Other Modules

The bit vectors returned from the IPrx and TCPrx modules are AND'ed together by the Aggregator module which passes the result to the Count module which increments the counters for the rules matched by a packet. When requested the user can press a button on the development board to pass the rules counts at that time out over a serial RS-232 interface.

6 Discussion

Chapter five described the implementation of a packet filter on an FPGA platform. This implementation has a few advantages and disadvantages. An advantage of this implementation is that data is read in every clock cycle and it is estimated that one clock cycle is required each for the Trie and PortBitVec modules to perform their matching operations.

Therefore it is estimated that 76 clock cycles are required to read in all the fields of interest (because a nybble of data arrives each clock cycle) and for the Trie module to return its result and then other clock cycle for the PortBitVec to return a result i.e. only one extra clock cycle is incurred by the filter rule set regardless of the size of the rule set.

However the time advantage of the filter is offset by the number of FPGA resources required to store the bit masks and the resources required by the PortBitVec module in matching the port numbers. The design described in chapter five was implemented with only ten filter rules as an indication of what its performance may be during operation and used 9% of the 4 input look up table (LUT) devices used for logic on the FPGA device.

Not all logic LUTs were used for rule checking however and by varying the rule set between best case and worst case it was found that there was a 3% difference in the number of LUTs required in each case. It is therefore feasible to scale up the sample rule set from 10 to possibly 100 rules thereby increasing the effectiveness and granularity of the rule set.

7 Conclusion

This paper describes a simple modularised packet filter implemented on an FPGA device that can be expected to operate at line speeds when placed on connection with a transmission speed of 100Mbps or slower. This filter can store and match packets against a variety of rules with the purpose of recording the number of times rules are matched and the counts can be returned to the user via serial interface.

Although currently the design parses only TCP packets the modular design of the filter means that support for other protocols can be added with relative ease to expand the versatility of the filter device.

References

[1] Lim Ming Hai. Overview of ethernet, 1997.

[2] Ji Li, Haiyang Liu, and Karen Sollins. Scalable packet classification using bit vector aggregating and folding. Technical report, MIT-LCS, 2003.

[3] Alastair Nottingham and Barry Irwin. Gpu packet classification using opencl: A consideration of viable classification methods. In *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, pages 160 – 169, 2009.

[4] V. Srinivasan, G. Vargheset, S. Suri, and M. Waldvogelg. Fast and scalable layer 4 switching. In *ACM SIGCOMM Computer Communication Review*, 1998.

[5] Rishard Stevens. *TCP/IP Illustrated*, volume One. Addison Wesley Longman, Inc., thirteen edition, 1999.

[6] David E. Taylor. Survey and taxonomy of packet classification techniques. ACM Computing Surveys, 2005.