Lightweight Bare-metal Stateful Firewall

Yihuan Xing
Dept. of Electrical & Computer Engineering
National University of Singapore
Singapore
xingyihuan@nus.edu.sg

Ford Long Wong
Information Assurance
DSO National Laboratories
Singapore
wfordlon@dso.org.sg

Akash Kumar
Dept. of Electrical & Computer Engineering
National University of Singapore
Singapore
akash@nus.edu.sg

***Abstract*— A firewall is a crucial security element in modern computer networks. This work investigates and demonstrates the implementation of a lightweight TCP/IP firewall in a bare-metal environment, on a commercial embedded ARM device. Compared to an implementation having an operating system (OS), using bare-metal design enables reduction of exposure to potential vulnerabilities in OS code, and provides a more dependable system. The implemented firewall provides both static and stateful filtering capabilities, and is configurable in a user-friendly way. As the architecture of the commercial hardware used was not available under closed source licensing, it was discovered through analysis at both hardware and software levels. Some challenges were encountered, and tools were developed to address these. The prototype is validated through functional testing in a controlled environment successfully.**

Keywords— firewall; bare-metal; ARM; stateful; encryptor

I. INTRODUCTION

A. Background

The introduction of computer networking has brought numerous benefits by enabling information sharing between different computers at both a local and global scale, but it had also suffered from malicious activities which compromise the safety of this communication medium. Vulnerabilities in protocols, applications, operating system code, and even unsafe user practices, are inherent parts of the problem.

A commonly used security mechanism is the firewall. This is often found at the point of entry between a private network and the Internet, such that all the traffic exchange between these two domains has to pass through the firewall.

The firewall's network protection functionality is often provided by software running on specialised embedded or general purpose computers. There are concerns over the security of such devices as the firewall software typically executes on an OS, where unnecessary lines of code and dynamic memories may introduce security loopholes. FPGA firewalls are potentially less susceptible to compromise, but it is harder to make these run more complex firewall logic.

Running firewall software in a bare-metal CPU configuration, where code is directly executed on the processor without any OS is an attractive alternative. Immediate benefits are higher performance and higher security, as the software does not need to pass through the OS's abstraction layers.

B. Contribution

This work demonstrates a lightweight firewall operating in a bare-metal environment on a commercial computer. The computer used was originally designed for an embedded Linux environment. It was analysed to discover the hardware architecture not available under closed source. The final product presents a bare-metal firewall with stateful packet inspection capabilities, as well as static packet filtering. This has also yielded tools to repurpose a commercial computer into a versatile device, with low source lines of code of about 9K.

II. REVIEW OF CONCEPTS

We assume familiarity with the OSI model. Firewall concepts are surveyed, to highlight the drawbacks of early-generation firewalls, and motivate stateful, and layer 2 firewalls.

A. Firewall Types

The firewall is a system providing traffic filtering between two network segments, according to user-defined security policy [3]. The firewall can be further classified into either a packet-filter or an application-level gateway. The packet filter operates between the network and transport layers, while the application-level gateway operates at the application level and is aware of application frames.

The early generation packet-filter firewall processes each packet individually, based on a fixed set of filtering rules. This posed a problem for protocols which relied on secondary connections for information exchange, such as the file transfer protocol (FTP). As the firewall is unable to distinguish such traffic, this secondary connection is rejected and results in a loss of connectivity by the applications. The stateful firewall concept was introduced to overcome some drawbacks of the static filter. This stateful firewall maintains a state machine for every connection that passes through it. The decision to permit or reject a packet is made based on connection state maintained by the firewall, on top of the packet information [4].

B. Layer 2 Firewalls

The typical firewalls operate from layer 1 to layer 3 OSI level. Layer 3 firewalls are usually used to replace routers at the edge of a trusted internal network. This results in the segregation of a single IP address space into two spaces. The fragmentation results in potential wastage of IP address, and increases the routing complexity and setup configuration.

978-1-4799-6474-1/14 $31.00 © 2014 IEEE
DOI 10.1109/PRDC.2014.15

The layer 2 firewall on the other hand does not segregate the network into two IP address spaces. This allows for a drop-in installation of the firewall, without the need to reconfigure any of the existing network devices as the firewall will appear transparent to them [3]. One downside is that the layer 2 firewall is harder to configure than a layer 3 one.

III. Problem Description

This section analyses our firewall requirements. The expected functionalities of the firewall were considered, with some simplifying assumptions for our prototype.

A. Physical Connection of Firewall to Network Environment

In our envisaged scenario, the firewall shall mediate between two networks and a crypto box (or IP encryptor). The firewall shall have four network interface ports, where one is connected to the trusted network, one to the untrusted network, and two connected to the crypto box as shown in Figure 1. The development of the encryptor is outside our scope, and it is assumed that encryption or decryption of packets is performed when they pass through, based on the direction. As the behaviour of the crypto box does not affect the essential operation of the firewall, for our test-bed the crypto box will be replaced with a direct pass-through point-to-point Ethernet link as shown in Figure 2. It is intended that the crypto box will appear transparent to the networked devices [1], so that the MAC addresses associated with the Ethernet frames processed remain the same when encrypted or decrypted.

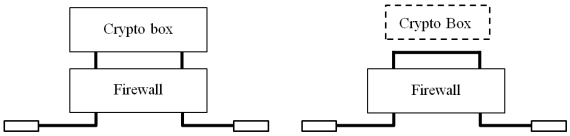

Fig 1–physical connection of firewall to network; Fig 2–simplified connection

B. Packet Filtering Rule

The default operating rule of the firewall will be to allow traffic to pass from the trusted to the untrusted segment via the crypto box for encryption. For a small number of packets which meet some static filtering criteria, the packets will be sent directly to the untrusted segment, by-passing the crypto box. The intuition underlying these rules are: packets generally need to be encrypted when they traverse the untrusted network towards the recipient, while there are some control and management packets which should not be encrypted (i.e. bypassed). The static filtering rules shall be reconfigurable by the user. Conversely, decryption will apply to traffic originating from the untrusted segment traversing towards the trusted segment generally, but they will also be subjected to more involved, stateful firewall filtering.

The stateful firewall should only allow traffic from the untrusted segment into the trusted segment only if the firewall had seen similar (i.e. like "--ctstate Established" in iptables) outgoing traffic. This stateful firewall filter needs to maintain a table of existing connections; adding and removing new entries if a new connection or connection termination is detected. The stateful filtering will also be required to remove connections from the table if the connection is inactive for a specified time.

C. User Interface for Firewall Configuration

A user-friendly and eventually secure way of reconfiguring the filtering rules of the firewall is required. Possibilities ranged from console-based to graphical user-interface based. Allowing configuration over the data network may open unforeseen entry points on the system, allowing malicious attacks from a remote location, therefore only the local serial port interface will be enabled for reconfiguration. A user application on a computer will provide the translation of user-defined rules in a GUI into a byte-oriented data stream for configuring the firewall via this serial port.

D. Network Layer Operation of Firewall

The transparency of the firewall can be achieved by cloning the MAC addresses between the ports connecting the two network segments, as shown in Figure 3. This results in a direct virtual logical link between Nodes A and B, despite being connected with two separate physical links.

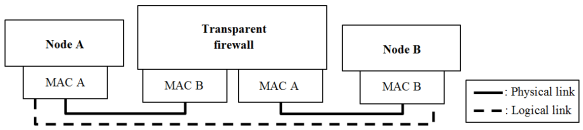

Fig 3 – Transparent firewall achieved by MAC address cloning

The transparent firewall assumes that there is a point-to-point connection beyond these two Ethernet interfaces with another OSI Layer 2 or higher network device. This assumption is important as the MAC cloning method described here will only work if the devices at the other end of the firewall's port only have one MAC address associated, i.e a point-to-point connection to a Layer 2 or above device.

E. Real-time system considerations

A real-time system describes a system where its logical correctness is based on the correctness of the outputs and their timeliness. The firewall should be in the form of a soft real-time system. Its response-time should be as fast as possible, but not restricted to absolute deadlines associated with catastrophic failures. The accuracy of the firewall on the other hand should be as accurate as possible, as an error in the firewall operation may allow malicious activity into the trusted segment.

The current prototype is not intended to perform any form of traffic shaping to provide Quality-of-Service. Traffic will be processed based on a first-in-first-out (FIFO) manner.

F. Development Environment

The bare-metal implementation requirement of the project calls for an integrated development environment (IDE) with appropriate compliers for the processor core used. The IDE should provide compilation and debugging capabilities for the target hardware. C is chosen as the implementation language.

IV. System Architecture

This section summarises the proposed hardware and software architecture of the firewall for implementation. The hardware constraints resulted in the need for a cyclic executive scheduler. The implementation of both static and stateful filters is discussed, describing the data structures and algorithm

required. The MAC address learning and cloning algorithm is presented, to achieve layer 2 transparent firewall operation.

A. Overview of proposed system architecture

The firewall is to be connected to the network via four Ethernet interfaces, while a host computer providing firewall configuration data is connected via a physical serial port. The Ethernet interface is implemented using existing commercial Ethernet controller hardware. This controller have two sets of buffers. The receiver buffer is used for storing Ethernet frames received from the network, while the transmit buffer is for storing frames to be sent into the network. The transmit and receive functionalities will be achieved by the main processing element reading and writing to these buffers.

The main processing element of the firewall is described in the firewall filtering process, shown in Figure 4. The Ethernet frames buffered by the Ethernet controllers are read into the main processing memory. This frame is then checked against the static and stateful filter rules list, and the frame is either dropped or rerouted to another Ethernet controller for transmission. The system provides a 1Hz clock, allowing the stateful filter to be time-aware and to remove inactive connections from its list.

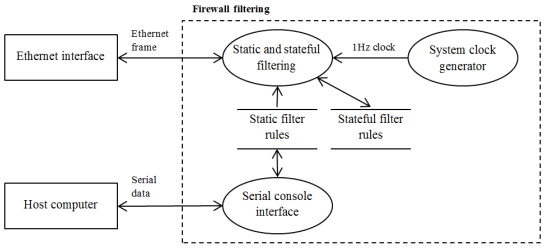

Fig 4 – Level 1 dataflow diagram of firewall filtering process

B. Hardware

We constrain the hardware for this work to an embedded computer, an Artila Matrix 514 shown in Figure 5.

Fig 5 – Artila Matrix 14 front view & top view

It is a commercial-off-the-shelf single-board computer with an ARM-based microcontroller. A Linux OS with version 2.6 kernel is bundled. It was chosen based on the low powered ARM processor, and it provides four Ethernet interfaces. The specifications of the Matrix 514 device can be consulted in [6].

The memory access times to both program code and data memory should be as quick as possible, for optimal performance. This can be achieved with preventing external bus contention by placing all program memory on internal ROM, and all data memory on internal SRAM. If external memories need to be accessed for the firewall application, the caching mechanism may be enabled to provide reduced memory access latency. The processor core may also be operated with normal ARM instructions, at the expense of lower code density over the reduced THUMB instruction set.

C. IPv4 packet de-capsulation

The behaviour of the firewall requires the inspection of IP address and TCP port numbers of the Ethernet frames received. The traditional design of a networked device calls for the use of an IP library to de-capsulate the Ethernet frames. The library is used to extract the source and destination IP addresses, and port numbers of different protocols, as well as the payload encapsulated by the carrier. As our scenario only expects IP traffic, this firewall will only need to provide filtering on IPv4 network traffic while other protocols can be subject to the default deny rule. The standardised protocol format meant that fields containing key information will be in the same byte offsets. This allows us to develop a lightweight IP support library to provide the necessary packet information extraction functionalities with the minimum code, instead of an IP library.

D. Cyclic executive scheduling

Executing in a bare-metal environment on a single-processor system presents a hurdle to implement concurrent processing at the software level so that multiple processes are seen to be executed simultaneously. A cooperative multi-tasking scheme in the form of cyclic executive will be necessary to provide scheduling between the different processes of the firewall software. This cyclic executive will guarantee zero resource conflicts, eliminating the need for any form of inter-process synchronisation. This scheduling method is represented by the state diagram illustrated in Figure 6.

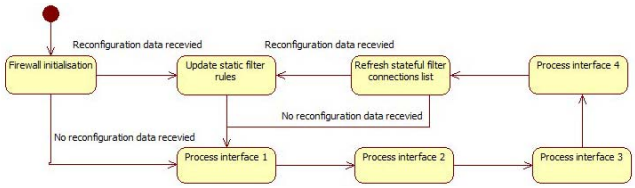

Fig 6 – State diagram depicting cyclic executive scheduling

After initialization, the firewall checks if any static filter configuration data is available on the serial input buffer. After updating the static filter rules, the firewall enters the cyclic state where it repetitively transitions between the states until powered down. In the process interface states, it checks for Ethernet frames stored in the receive buffer of the associated Ethernet controllers. The buffered Ethernet frame is processed by checking against the static and stateful filtering rules, and forwarded to the corresponding output Ethernet interfaces.

The firewall then transits to the next state, where it performs a similar operation on a different Ethernet interface. If no Ethernet frame is ready in the receive buffer, the processing for the Ethernet interface is skipped and transited to the next state. After processing all the four Ethernet interfaces, the relative timestamps in the stateful filters and logical timer are refreshed if the logical timer is near the overflow state. The receive buffer of the serial interface is also checked for updated static filtering configuration data. This new data is read and updated into the static filters of the firewall, before resuming the cyclic state of the firewall.

E. Static filter

The static filtering of the firewall is used to determine which output interface the received Ethernet frame should be sent to. If the Ethernet frame matches the relevant description in the static filter rules list, the Ethernet frame is allowed to by-pass the crypto box. This frame is then accordingly sent directly into the trusted or untrusted network. The comparison between the received Ethernet frame and rules list is done using a sequential list search. The search traverses down the stored rules list and returns a positive result if an exact match is found, else a negative result is returned.

The static filter rules list do not constantly change over time, except during the rare user-initiated reconfiguration. The rules list can be implemented with statically allocated memory such as an array in C. It will comprise of the following four fields per rule entry: source IP address, source port number, destination IP address, and destination port number.

We assume a manageable number of static filter rules. An array will be declared as lookup table for TCP stateful filter. An example is illustrated in Figure 7, showing two filter rules.

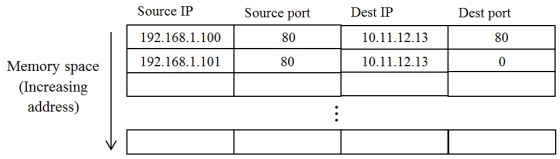

Fig 7 – Lookup table for static filtering

F. Re-configurability of static filter

We developed a communication protocol to be used for the firewall's configuration. A two-way handshake using ASCII encoded commands will be used to place the firewall into a mode to receive the configuration data. Upon successful reception of the configuration data, the firewall will acknowledge the transfer and close the connection before updating the static filter rules list. In future versions, cryptographic authentication methods could be implemented to provide more secured access.

The host computer will always send a fixed amount of bytes determined by the maximum filter entries in the firewall. The fields are sent to the firewall with lowest byte first in the format shown in Figure 8. A GUI program running on a host computer was developed, that generates the necessary data stream compliant to the data format described, and handles the handshaking for the data transfer.

Byte	0-3	4-7	8-11	12-15	16-19	
Field	Source IP	Source port	Dest IP	Dest port	Source IP	…
	TCP entry 1				TCP 2	

Fig 8 – Data format of firewall configuration data sent over serial interface

G. Stateful filter

The stateful filter is responsible for allowing traffic from the untrusted network into the trusted network, only if similar traffic has been seen outgoing to the untrusted network. The stateful filter on the other hand allows all traffic from the trusted network to traverse into the untrusted network, and stores this event in its active connections list. The stateful filter monitors the TCP packet for the FIN-ACK flag, which then removes the associate entry from the active connections list.

Unlike the static filter where its filter list does not constantly change, the stateful filter's list may change as frequent as every cycle of the cyclic executive. This frequently changing filter list will require a data structure capable of dynamic assignment with low processing overhead for data insertion or deletion. The linked list data structure meets the described needs and can provide possible speedup on certain scenarios.

This linked list can be constructed by statically allocating two memory arrays on the firewall. The first array is used to store the IP address of the stateful entry, and serves as an index to provide entry point to the main data array. The second array stores the port number and timestamp of the last connection made with the associated IP address in the index table.

An example of an active connection to destination IP address 10.11.12.13 on destination ports 5000 and 5001 is shown in Figure 9. When searching the stateful filter, the firewall first performs a sequential search in the index array. The firewall then traverses down the associated linked list looking for a match in the port number. The linked list search returns negative if no match is found in either stage.

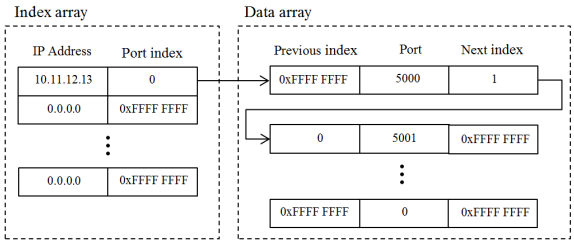

Fig 9 – Example of linked list with one IP address and two ports

When a new connection is established, the new entry is always appended at the front of the linked list. This exploits temporal locality of the data access [5], assuming that the new connection established here will most likely be accessed soon, as compared to connections established earlier. This strategy reduces the number of traverses required before convergence, leading to less processing time for the search.

The entries in the stateful filter table are removed if no active connection is seen for more than a user-specified time (e.g. 3,600 seconds). The firewall also inspects all the TCP packets for the FIN-ACK flag, which signals the end of current TCP transmission session. Upon detection of such packets, the firewall removes the associated entry from the stateful filter.

H. MAC address learning

It is useful for the firewall to appear transparent to the attached network device; at other times the MAC bindings can be hard-coded before use. A MAC address cloning method was developed and implemented. This was achieved by inspecting the Ethernet frames that the Ethernet interface sees immediately after a successful physical media negotiation.

V. Implementation Issues

Some significant challenges were faced during implementation. Solutions and workarounds were developed.

A. Closed source of hardware design

Development in bare-metal environment required low-level drivers for access to the hardware peripherals. This required prior knowledge of the logical and physical connections of such peripherals. However, the hardware schematic of the Matrix 514 device is closed source. On the other hand, the Linux OS used by the Matrix 514 is open source, and analysing this provided insights into the hardware architecture. Also, the device was physically taken apart for analysis at the component level. The functionalities of the integrated circuit modules were discovered through searching public datasheets. Spatial locality between components also provided clues. It was discovered that five peripherals are mapped onto the external address/data bus of the microcontroller system, which are three DM9000, one SDRAM, and one NOR flash.

B. Lack of in-circuit emulator/debugger

The Matrix 514 uses an ARM9-based microcontroller with in-circuit emulation accessible over JTAG. A compatible Segger J-Link JTAG emulator used was unable to detect the microcontroller, returning no valid devices found on the boundary scan. Some configuration may have disabled the JTAG functionality. Alternatively, the serial port was used to periodically print the firewall state for debugging, implemented as a blocking call. But this complicated the development of advanced processor mechanism, such as direct memory access (DMA) and interrupts, leading to data transfer between Ethernet interfaces not being accelerated, and high processor overhead. The Ethernet interfaces were polled periodically instead, but increasing latency in responding to incoming data.

C. Booting with bare-metal program

The factory state of the Matrix 514 only allowed the microcontroller to boot from the external NOR flash. However, when a high logic is applied to the BMS, the microcontroller boots from the internal ROM instead of external flash. When the microcontroller does not find any valid program during this boot, it would expect to receive firmware upgrade over the debug serial port. This built-in firmware upgrade mechanism takes in a binary executable file, stores it in the internal SRAM, and executes the binary file like a normal program from the SRAM. This method is restricted to a program memory size of less than 13Kbytes. We adapted this mechanism to load our program.

D. Volatile storage of bare-metal program

It is desirable to store the bare-metal program permanently in the reprogrammable external NOR flash, as compared to the one-time-programmable internal ROM of the microcontroller. So, another bare-metal program was developed, which reads a binary file via the serial port and writes it into the external NOR flash. However, while it was possible for the device to boot this way a simple program we developed, there were issues with getting it to boot the firewall program. This was traced to an environment configuration of the IDE used, which required a different linker description if the bare-metal were stored in external flash. This problem is only partially resolved.

VI. Results

The results of validating the firewall implemented are presented. The firewall was also subjected to benchmarking to provide a performance baseline.

A. Bare-metal program

The firewall bare-metal program was successfully implemented in Keil µVision 4.7 with ARM MDK-Lite, free license with code and data size restriction of 32Kbytes. The compiler produced a binary executable file. The size of the compiled program is 8,492bytes, below the maximum 13Kbytes allowed by the firmware upgrade mechanism. The firewall was verified to be up by observing for messages on the debug serial port of the Matrix 514 device. Compared to the Linux kernel OS (version 2.6) which has 4M SLOC, the bare-metal firewall program has only around 9K SLOC.

B. MAC address learning

The MAC address learning functionality was tested with a consumer-class router and a laptop computer. The firewall bare-metal program was uploaded, and the MAC address learning procedure was allowed to run. The procedure was validated to be able to discover the devices' MAC addresses.

C. Reconfiguration capability

Reconfigurability was achieved through the development of a Windows-based application using Microsoft Visual Studio 2010 C#. The developed application provided an all-in-one tool to upload the bare-metal program into the device, upload static filter rules, and visually display data received on the serial port.

D. Static filter functionality

The functionality of the static filter was evaluated using different test scenarios:

- Allow all traffic to bypass crypto box
- Allow some traffic to bypass crypto box
- Allow no traffic to bypass crypto box

The static filter was tested with direct connections to one packet generator, and one packet sniffer, as shown in Fig. 10.

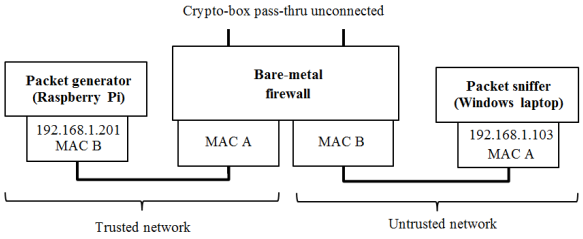

Fig 10 – Test setup for static filtering

The packet generator was placed on the trusted network side, so that the stateful filter did not block any of the packets from propagating towards the packet sniffer. The packet generator was realised using a Raspberry Pi embedded Linux

computer executing Mausezahn, a network traffic generator. This setup provided the flexibility of specifying the source and destination IP addresses, and port numbers of the TCP packets. The packet sniffer was realised using Wireshark.

The packet generator was configured to generate a burst of TCP packets. The different static filter rules were then configured into the firewall, and the packets received on the packet sniffer is noted and consolidated in Table 1.

Test scenario	Static filter rules (IP, Port)	Packets received on sniffer
Allow all traffic to bypass crypto box	Source: 255.255.255.255, 0 Dest: 255.255.255.255, 0	TCP with source/destination ports 81,82,83,84,85
Allow some traffic to bypass crypto box	Source: 192.168.1.201, 82 Dest IP: 192.168.1.103, 82	TCP with source/destination port 82
Allow no traffic to bypass crypto box	No entry	No packets received

Table 1 – Summary of static filter test results

The test results showed that when no traffic is allowed to bypass the crypto box, the packet sniffer does not receive any packets from the generator. The test results also showed that it was able to selectively pass packets when filtering criteria was met, as well as allow all traffic to bypass the crypto box. The pass thru cable was unconnected to help distinguish bypassed and non-bypassed packets.

E. Stateful filter functionality

The functionality of the stateful filter was similarly evaluated like the static filter test, using the following test scenarios:

- Access TCP port in untrusted network
- Access test computer's TCP port from untrusted network at cold state
- Access test computer's TCP port from untrusted network after initiating an outgoing connection
- Access test computer's TCP port with a timed out entry

The stateful filter was tested with almost the same hardware setup as in Figure 10, with pass thru cable connected. The packet generator and sniffer was swapped, such that packets were sent towards the trusted network. In addition to packet sniffing, the test computer now performed packet generation to insert entries in the stateful filter which allow traffic in from the untrusted network. The packet generation was realised using Ostinato, a packet generator software.

The static filter was configured to permit all traffic to bypass the crypto box, so that this test was independent of static filtering. The packet generator was configured to generate the similar packet burst, as before. The stateful filter was then subjected to four test scenarios, and the response of these test scenarios were recorded and shown in Table 2.

The test results showed that the stateful filter was able to block traffic which was not included in the active connections list. The filter was able to append new established TCP sessions into this list, which allowed previously blocked traffic to now enter into the trusted network. Finally, the filter was able to identify and remove inactive connections from the list, in accordance to the proposed design.

Test scenario	Events observed
Access test computer's TCP port 84 from untrusted network at cold state	Blocked IP 192.168.1.201, port 84
Access port 84 in untrusted network from test computer	Added IP 192.168.1.201, port 84
Access test computer's TCP port 84 from untrusted network after initiating an outgoing connection	Received TCP packet with source/destination ports 84 by sniffer
Access test computer's TCP port 84 with a timed out entry	Removed IP 192.168.1.201, port 84

Table 2 – Summary of stateful filter test results

F. Network performance

A network benchmark tool, LANBench is used to determine the firewall performance in bare-metal and original Artila Linux environments and results are consolidated in Table 3. The small disparity in the results suggests that if DMA and interrupts are implemented, the bare-metal case may even produce performance superior to the OS-based environment.

Test case (speed in kbits/sec)	Artila Linux	Bare-metal
Max speed (operated as Ethernet bridge)	2,487	2,141
Average send speed	2,278	1,726
Average receive speed	2,316	1,583

Table 3 – Summary of firewall switching performance

VII. CONCLUSIONS

The work aims at the development of a lightweight firewall in bare-metal using an embedded Artila Matrix 514 ARM-based computer. A cyclic executive scheduler was developed, to overcome the lack of native concurrent processing support. The behaviour and algorithms of the static filter, stateful filters, and MAC address learning were developed.

The functionality of the firewall was validated with a series of test cases, targeting the different software modules. The tests revealed a successful implementation of the firewall based on the specification. The bare-metal program's SLOC at 9K compares very favourably with any Linux OS-based solution, which would require at least 4M SLOC, representing a great reduction in complexity and possible vulnerabilities in code.

Possible future work include: setting up the JTAG interface to enable better debugging, development of DMA and interrupts, implementing more extensive firewall policies, performance optimisation, and formal analysis of the design.

REFERENCES

[1] M. Gonçalves, *Firewalls : a complete guide / Marcus Gonçalves*: New York : McGraw-Hill, c2000., 2000.

[2] J. F. Kurose and K. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*, 5 ed.: Addison-Wesley Publishing Co., 2010.

[3] C. Douligeris, *Network security: current status and future directions*. Hoboken, N.J: Wiley, 2007.

[4] M. G. Gouda and A. X. Liu, "A model of stateful firewalls and its properties," in *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, 2005, pp. 128-137.

[5] M. Frantzen, F. Kerschbaum, E. E. Schultz, and S. Fahmy, "A Framework for Understanding Vulnerabilities in Firewalls Using a Dataflow Model of Firewall Internals," *Computers & Security,* vol. 20, pp. 263-270, 5/1/ 2001.

[6]Artila, *Artila Matrix 514 Data Sheet*, http://www.artila.com/docs/Matrix-514/ Matrix-514%20Data%20Sheet.