NetFPGA DNS Server

Richard W. Wallen

***Abstract*— The Domain Name System (DNS) is a distributed database that allows users to resolve host names to Internet Protocol (IP) addresses. DNS allowed the Internet to substantially grow to form the Internet as we know today. The backbone of the Domain Name System consists of a hierarchical structure of DNS servers. A DNS server residing in hardware would allow for optimizations that could not be explored in a typical server setup.**

1 INTRODUCTION

Designing a DNS server in hardware started with researching and understanding DNS and how its name servers worked. DNS packets sit at the application layer, though it is often not a network application one typically thinks about. One typical DNS function is to resolve domain names to IP addresses.

Traditionally, DNS servers utilize software running on some operating system, which performs computations on a general-purpose piece of hardware. Currently, this architecture remains relatively unchanged and works well. However, working well does not mean that other implementations should not be explored. A DNS server running on an application specific integrated circuit (ASIC) device can take advantage of hardware specifically designed for its own architectural needs, which could allow faster lookup times, yield higher throughput, provide quicker resolution of domain names, and make DNS servers more robust.

Field Programmable Gate Arrays (FPGA's) allows for inexpensive prototyping of ASIC devices. The NetFPGA is designed specifically for network applications. The NetFPGA has been used extensively for teaching lower levels of the network stack and in academic research. The NetFPGA makes for a perfect device to use for designing a hardware DNS server. The NetThreads compiler project is a NetFPGA project created to ease prototyping and coding on the NetFPGA device.

2 Domain Name System (DNS)

Prior to the Domain Name System (DNS) a file was maintained by the Network Information Center that contained a list of host names mapped to addresses. During the start of the Internet, this system worked well since the number of hosts was relatively small and updates to the list were relatively infrequent. When a new host would join, the host would request to be added to the list, the list would be updated, and the updated list would be sent out to all the hosts. As increasing amount of hosts began to connect to the Internet, the process of updating the list proved inadequately slow and the list grew too large. A new system was required to accommodate the frequency of changes and the growing number of hosts. A distributed hierarchical database was born. That database was the Domain Name System or DNS.

DNS distributes its workload to a hierarchical system of servers, each of which has a specific realm of knowledge. Those realms are better known as zones or domains.

At the top, there are general domains and the DNS servers, also called name servers, in charge of those domains. Those domain or zones can be further divided into sub-domains with DNS servers in charge of those domains, which can be divided on and so forth. That division occurs with the structure of the domain name space. Updates need only happen to a DNS server in charge of the zone where modifications have occurred.

Finding a host's address happens by requesting it from the DNS server in charge of the zone that the host belongs. Requests can occur at any DNS server; where the DNS server will either answer the request or point to a DNS server better suited to potentially answer the request. Finding a better-suited DNS server to handle the request occurs by either going to a more generic zone's name server or a more specific zone's name server. That process occurs until the DNS server of that zone is found. A DNS server can perform this process, if the server provides that service.

However, finding addresses is not the only function of the Domain Name System. There are a variety of queries that can be answered by this system. [RFC 1034] serves as a description of functions of DNS. It introduces DNS structure and general information of best practices on: 1)resource records (the information on a DNS server), 2)DNS message protocol (the way queries and responses are communicated), and 3)master/zone file format (the way information is loaded to the DNS server). [RFC 1035] details the formats of several resource records, DNS message protocol, and master/zone file format. [RFC 1033] provides a guideline of best practices for administering a zone and name server.

DNS has standards for: the formats for resource data (the answers provided from DNS servers), methods for querying the DNS servers (DNS message protocol), and methods for name servers to refresh local data from foreign name servers (Master files or Zone files). This includes how a system administrator needs to format their master files or zone files that transfers to foreign servers. (Master files are text files that are read into the local name server [RFC 1034].) These standards were written in such a way as to accommodate future changes that needed to occur.

2.1 Domain Name Space

The domain name space is a hierarchical structure where each level, known as labels, gets separated by a '.' (pronounced 'dot'). Domain names (DN) are restricted to a maximum of 255 characters in total length. A label has a maximum character length of 63. In domain names, case is not relevant, even though it is preserved. Take for instance the Fully Qualified Domain Names (FQDNs) : *missouri.edu.* and *MISSOURI.EDU.* they both will resolve to the same IP address since character case is insignificant.

The hierarchy reads from precise to general, or the prefix as most specific and the suffix and least specific. Each label depicts a sub-domain of a higher level and labels must be unique in that particular sub-domain space. This means that members of a sub-domain, also referred to as siblings, cannot have the same label

Take for example the FQDN: *missouri.edu.* the label *edu* is a generic top level domain (gTLD) specifying that the domain name is part of an educational institution's domain name. The label *missouri* specifies it belongs to the Missouri domain within the gTLD domain *edu.* Each dot in an FQDN segregates the labels and the dot at the end represents the root zone (no it is not a period at the end of a sentence).

This hierarchical structure fits that of a tree. A dot typically represents the root of the tree and each generic top level domain would be a child of the root. Furthermore, each sub-domain of a gTLD are children of that gTLD. Subsequently, the reverse order of a domain name orders from parent to child, the dot or root is assumed if not present at the end of the domain name. (A shorthand notation of a domain can be specified when working in a particular zone. For example, if we are in the namespace *missouri.edu*. we can query the domain name *cs.missouri.edu.* by only using *cs* the *missouri.edu.* would be appended and checked to see if that sub-domain exists if the name server allows such shorthand notation. There is no support for this shorthand notation in the implementation of this project.) Also, since the domain name space can be represented in a tree structure a Trie structure can be utilized.

2.1.1 Zones

A zone defines the contents of a contiguous section of the domain space [RFC 1033]. System administrators define the name space boundary of a particular zone. A master file contains resource records for a particular zone and is loaded into a name server.

2.2 Name Servers or DNS Servers

Name servers or DNS servers preside over a zone or zones and possess authority and knowledge of that zone or zones. A name server is only part of this distributed database and provides the ability to allow the system to be updated quickly and independently by a local administrator. Much like the domain name space, the name servers are hierarchical.

The name servers are arranged based upon the domain name space that defines the zone or zones and can be divided further by additional DNS servers managing zones underneath. The root servers are servers that sit at the generic top level domains and provide information about those zones and major name servers within those zones. The root servers are updated less frequently in comparison to other DNS servers. A name server should maintain a list of these root servers and can obtain a current list through any of the root servers. The implementation in this paper assumes that that list has been received prior to loading the DNS server instructions.

Name servers must be able to handle the workload and can do so by means of redundant name servers. These name servers may have information cached about other zones and knowledge of that information is non-authoritative. A name server can only be authoritative over the zones, or section of domain space, it presides over. These name servers handle queries from hosts and provides answers given the resource records loaded from master files.

2.3 Master Files

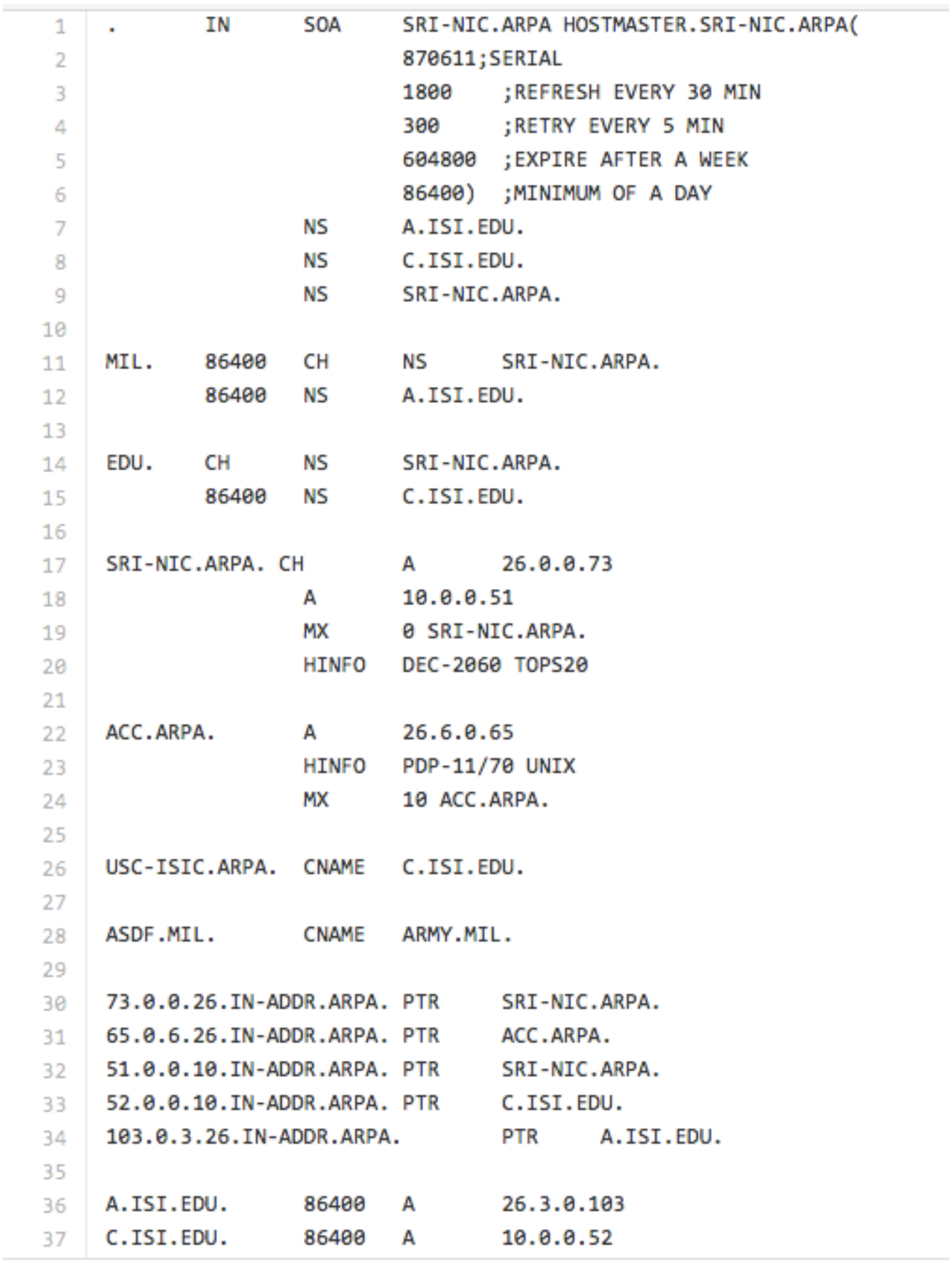
Master files are used to load resource records to the name server. These resource records contain information about the domain name space in which the DNS server is the authority. Master files are updated by the local system administer as the zone changes and are used to refresh the name server(s).

The basic format of the master file is based upon the [RFC 1034], [RFC 1035]. Several modifications were made to the format. Accommodating the shorthand notation for multiple resource records had to be made. Also, the order in which the resource records are stated needed to be defined.

In shorthand notation of the master file, the name is stated first and each new line that contains a new name is part of the previous name. Keeping things simple, each line either begins with a name or a tab. Therefore, the server does not have to determine whether the file is indeed a name or a resource record.

Staying true with keeping things simple, the format specifying resource records needed a stringent definition. Formatting the order in which the network type and cache time for a particular resource record needed specification. The format is listed below as well as an example zone file.

- Comments are denoted with semicolons, ‘;’
- Records typically are contained in one line unless surrounded by parenthesis ‘(‘ and ‘)’.
- TTL and CLASS are optional
- Owner name is listed at the beginning of the record
- If owner name is not specified at the beginning then the prior owner name is used
- White space delimited



```
.       IN      SOA     SRI-NIC.ARPA HOSTMASTER.SRI-NIC.ARPA(
                        870611;SERIAL
                        1800    ;REFRESH EVERY 30 MIN
                        300     ;RETRY EVERY 5 MIN
                        604800  ;EXPIRE AFTER A WEEK
                        86400)  ;MINIMUM OF A DAY
                NS      A.ISI.EDU.
                NS      C.ISI.EDU.
                NS      SRI-NIC.ARPA.

MIL.    86400   CH      NS      SRI-NIC.ARPA.
        86400   NS      A.ISI.EDU.

EDU.    CH      NS      SRI-NIC.ARPA.
        86400   NS      C.ISI.EDU.

SRI-NIC.ARPA. CH        A       26.0.0.73
                A       10.0.0.51
                MX      0 SRI-NIC.ARPA.
                HINFO   DEC-2060 TOPS20

ACC.ARPA.       A       26.6.0.65
                HINFO   PDP-11/70 UNIX
                MX      10 ACC.ARPA.

USC-ISIC.ARPA.  CNAME   C.ISI.EDU.

ASDF.MIL.       CNAME   ARMY.MIL.

73.0.0.26.IN-ADDR.ARPA. PTR     SRI-NIC.ARPA.
65.0.6.26.IN-ADDR.ARPA. PTR     ACC.ARPA.
51.0.0.10.IN-ADDR.ARPA. PTR     SRI-NIC.ARPA.
52.0.0.10.IN-ADDR.ARPA. PTR     C.ISI.EDU.
103.0.3.26.IN-ADDR.ARPA.        PTR     A.ISI.EDU.

A.ISI.EDU.      86400   A       26.3.0.103
C.ISI.EDU.      86400   A       10.0.0.52
```

Figure 1 Example Master File

2.4 Resource Records

Resource records provide the information contained within a zone. A resource record pertains to a particular domain name within a zone. The general form of a resource record can be seen in the figure below.

2.4.1 Resource Record Format

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NAME (MULTIPLE OCTETS)															
TYPE															
CLASS															
TTL															
RDLENGTH															
RDATA (MULTIPLE OCTETS)															

Figure 2 General Form of Resource Records

2.4.2 Name

Name refers to the domain name of the node in which the resource records pertains. The name is variable in length; however labels are still limited to a maximum of 63 bytes and the overall length has a maximum of 255 bytes.

2.4.3 Type and Corresponding RDATA

The type field is represented by an unsigned 16-bit integer or is 2 bytes. The number corresponds a specific kind of resource record, which will denote what is contained in the RDATA section. Listed below are some of the more common resource record types, which are the types chosen to be used in the implementation. There are other types; however, some are experimental, some have been obsoleted, and some are less commonly used. The types used were defined in the papers [RFC 1034] and [RFC 1035] with exception to AAAA records, which came from [RFC 3596]. AAAA records are important with greater adoption on IPv6 since AAAA records are the records of IPv6 addresses.

2.4.3.1 Type A and RDATA

16-bit Type Value: 1
Host address

ADDRESS

Figure 3 Type A RDATA section

2.4.3.1.1 ADDRESS

ADDRESS is a 32-bit IPv4 address that corresponds to the domain name in the name section. RFC 1034 references that this address section could be dependent on the class of the record; however, RFC 1035 determines this type of record to be internet specific.

2.4.3.2 Type NS and RDATA

16-bit Type Value: 2
An authority name server

NSDNAME

Figure 4 Type NS RDATA section

2.4.3.2.1 NSDNAME

NSDNAME is the domain name of a name server that would be a reference to a name server that would be outside the boundary of the current zone’s boundary. If the query turns out to be outside the current zone’s boundary and a name server record is encountered, the name server record would be put into the authority record section and any addresses associated would be copied into the additional records section.

2.4.3.3 Type CNAME and RDATA

16-bit Type Value: 5
Canonical name

CNAME

Figure 5 Type CNAME RDATA section

2.4.3.3.1 CNAME

CNAME is the proper domain name, or canonical name, of the domain name that appears in the name section. The domain name appearing in the name section is an alias. In the situation where the query type is not requesting the CNAME type, the DNS server would subsequently fire a search for the query type using the CNAME to find the corresponding resource record(s) and put that record(s) into the addition record section.

2.4.3.4 Type SOA and RDATA

16-bit Type Value: 6
Statement of authority

MNAME
RNAME
SERIAL
REFRESH
RETRY
EXPIRE
MINIMUM

Figure 6 Type SOA RDATA section

2.4.3.4.1 MNAME

MNAME is the domain name of the primary source of data from this zone used for refreshing redundant name servers.

2.4.3.4.2 RNAME

RNAME is the domain name of the mailbox of the system administrator of this zone

2.4.3.4.3 SERIAL

SERIAL is a 32-bit unsigned integer that is the version number of the original source for the zone. This is used for the redundant name servers.

2.4.3.4.4 REFRESH

REFRESH is a 32-bit time interval used for redundant name servers for checking the SERIAL number to determine whether the zone needs to be refreshed

2.4.3.4.5 RETRY

RETRY is a 32-bit time interval determining the time between attempts to refresh the zones. This is used in redundant name servers in case the check fails when the refresh time interval elapsed.

2.4.3.4.6 EXPIRE

EXPIRE is a 32-bit time interval determining the length of time that a zone is authoritative. If the retry attempts fail, and the expiration time limit has elapsed, the redundant name server determines its zone is obsolete and discards its files and must receive a zone transfer.

2.4.3.4.7 MINIMUM

MINIMUM is 32-bit unsigned integer that is the minimum TTL for any resource record in the zone.

2.4.3.5 Type PTR and RDATA

16-bit Type Value: 12
Pointer name

PTRDNAME

Figure 7 Type PTR RDATA section

2.4.3.5.1 PTRDNAME

PTRDNAME is a domain name of the record. The name in this case is its address in reverse order of octets with the '.IN-ADDR.ARPA.' domain appended at the end. Its primary usage is for reverse lookups of addresses to hosts.

2.4.3.6 Type MX and RDATA

16-bit Type Value: 15
Mail Exchange

PREFERENCE
EXCHANGE

Figure 8 Type MX RDATA section

2.4.3.6.1 PREFERENCE

PREFERENCE is a 16-bit unsigned integer that determines the priority of the resource record. Lower numbers have priority.

2.4.3.6.2 EXCHANGE

EXCHANGE is a domain name of a server that is able to serve as a mail exchange server for the name stated in the name section.

2.4.3.7 Type AAAA and RDATA

16-bit Type Value: 28
128 bit IPv6 address for host

ADDRESS

Figure 9 Type AAAA RDATA section

2.4.3.7.1 ADDRESS

ADDRESS, much like the address for an A resource record, is a 128-bit IPv6 address that corresponds to the domain name in the name section.

2.4.4 TTL (Time To Live)

Time to live is a 32 bit unsigned integer which defines the amount of time that record is valid before it needs to be refreshed by the name server. That unit of time is defined in seconds.

2.4.5 Class

Class determines the protocol family that the resource record pertains. The implementation of the DNS server focused on the internet class, since it was the most common. The other classes are referenced to acknowledge their existence.

2.4.5.1 IN

16-bit Class Value: 1
Internet

2.4.5.2 CS (Obsolete)

16-bit Class Value: 2
CSNET

2.4.5.3 CH

16-bit Class Value: 3
Chaos

2.4.5.4 HS

16-bit Class Value: 4
Hesiod

2.4.6 RDLENGTH (Record Data Length)

This is an unsigned 16-bit integer of the length of the record data content. Since the type of the resource record dictates the content of the record data, and there are records that can be variable in length, the length is stated.

2.4.7 RDATA (Record Data)

Record data is the content of the record and its type dictates the information contained within the record.

2.5 Resolvers

Hosts interact with DNS servers through use of a resolver. Resolvers have a client server relationship with DNS servers, where the resolver would be the client. Resolvers act as a way for users to extract information from name servers through queries. A resolver will query a DNS server to extract the resource record on domain names. Resolvers and a DNS server communicate through the DNS protocol. The DNS protocol sets the way DNS packets are formed.

2.6 DNS Message Protocol

The general form of a DNS message protocol is provided in Figure 10. DNS sits in the application layer of the Open System Interconnection (OSI) model. DNS packets are typically enclosed in User Datagram Protocol (UDP) packets, a transport layer protocol. However, with zone transfers, the packets are enclosed within Transmission Control Protocol (TCP) packets, a more reliable transport layer protocol in which dropped packets are re-sent. This is due to the importance of the integrity of a zone transfer and its larger size.

HEADER
QUERY
ANSWER
AUTHORITY
ADDITIONAL

Figure 10 Overview of a DNS Message or Packet

2.6.1 DNS Header

The DNS header describes what is contained in the DNS packet. The header distinguishes queries and responses. It tells what the DNS server capabilities are. Also, it determines the number of queries and resource records contained within the packet.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ID															
QR	OPCODE				AA	TC	RD	RA	Z			RCODE			
QDCOUNT															
ANCOUNT															
NSCOUNT															
ARCOUNT															

Figure 11 DNS Header

2.6.1.1 ID

ID is a 16-bit unsigned integer specifying a identification number.

2.6.1.2 QR

QR is a 1-bit field that determines whether the DNS packet is a query or a response (answer).

- 0 is a query
- 1 is a response

2.6.1.3 OPCODE

OPCODE is a 4-bit field that determines the type of query. It is set by the resolver to notify the DNS server of the type of query being asked.

- 0 is a standard query
- 1 is an inverse query (not implemented)
- 2 is a server status request (not implemented)
- 3-15 has been set aside for future use

2.6.1.4 AA

AA is a 1-bit field that stands for Authoritative Answer. It is set by the DNS server to notify the resolver whether or not the answer is an authoritative one.

- 0 for non-authoritative
- 1 for authoritative

2.6.1.5 TC

TC is a 1-bit field that stands for Truncation. It notifies whether the message has been truncated due to length and that another packet will finish the message.

- 0 for not truncated
- 1 for truncated

2.6.1.6 RD

RD is a 1-bit field that stands for Recursion Desired. It is set by the client to ask the DNS server to perform a recursive query.

- 0 for non-recursive query
- 1 for recursive query is desired

2.6.1.7 RA

RA is a 1-bit field that stands for Recursion Allowed. It is set by the server to notify the resolver whether recursive queries are permitted.

- 0 for not allowed
- 1 for allowed

2.6.1.8 Z

Z is a 3-bit field set aside for future use.

2.6.1.9 RCODE

RCODE is a 4-bit field that stands for response code. It is set by the DNS server to notify the resolver of the status of the response.

- 0 for no error
- 1 for format error notifying the resolver it was unable to interpret the query
- 2 for server error notifying the resolver that the server has a problem and is unable to answer the query
- 3 for name error, which if the server's response is authoritative, means that the domain name does not exist
- 4 for not implemented, meaning that the query performed is not supported.
- 5 for refused

- 6-15 has been set aside for future use

2.6.1.10 QDCOUNT

QDCOUNT is an unsigned 16-bit integer, which stands for the number of entries in the query section.

2.6.1.11 ANCOUNT

ANCOUNT is an unsigned 16-bit integer representative of the number of entries in the answer section.

2.6.1.12 NSCOUNT

NSCOUNT is an unsigned 16-bit integer representative of the number of name server resource records in the authority records section.

2.6.1.13 ARCOUNT

ARCOUNT is an unsigned 16-bit integer representative of the number of resource records in the additional records section.

2.6.2 DNS Query

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
QNAME (MULTIPLE OCTETS)															
QTYPE															
QCLASS															

Figure 12 DNS Query (QNAME, QTYPE, and QCLASS repeated QDCOUNT times)

2.6.2.1 QNAME (Query name)

Query names are domain names for which the resolver requests the DNS server to supply a corresponding resource record(s). However, in these domain names each label is separated by an octet (unsigned 8 bit integer) that tells the length of the next label.

2.6.2.2 QTYPE (Query Types)

Query types notify the DNS server of the type of resource record being requested. There are additional types, as query type is a superset of type, meaning that they contain all the types with additional types specific for queries. These extra types are listed below; however, are not supported in the implementation of a DNS server in order to keep the queries and responses simple and easy.

2.6.2.2.1 AXFR

16-bit QTYPE Value: 252
AXFR represents a request for the entire zone transfer.

2.6.2.2.2 MAILB

16-bit QTYPE Value: 253
MAILB represents a request for mailbox related records (MB, MG, or MR).

2.6.2.2.3 MAILA

16-bit QTYPE Value: 254
MAILA represents a request for mail agent resource record (obsolete see MX).

2.6.2.2.4 *

16-bit QTYPE Value: 255
* represents a request for all records.

2.6.2.3 Query Class

Query class notifies the DNS server on the class of resource record being requested. Much like query types, query class is a superset of class. The only extra class is the wildcard '*' meant for any class. The wildcard query class is not supported in the implementation of a DNS server in order to keep the queries and responses simple and easy.

2.6.2.3.1 *

16-bit QCLASS Value: 255
* represents a request for All Classes.

3 NetFPGA

The NetFPGA is an Open Source network specific Field Programmable Gate Array (FPGA) which allows for quickly designing and prototyping network hardware. FPGAs are used to quickly design, prototype, and test an ASIC using a Hardware Description Language (HDL) such as Verilog. The NetFPGA uses Verilog to program the FPGA and is capable of line rate packet processing. The NetFPGA has been used in research to develop and test network designs.

The first NetFPGA V1 was developed at Stanford in 2001 as a teaching tool for learning the lower levels of the network stack, the link, and physical layers. However, the first generation device was redesigned in order to fix several issues. These issues with the first generation device include: 4x10Mbps ethernet interfaces (slow ethernet interfaces), it lacked CPU, it used outdated FPGAs and development tools, and it needed to be manually assembled.

3.1 NetFPGA 1G

In 2004 the NetFPGA 1G replaced the NetFPGA V1. The NetFPGA 1G has 4x1Gbps ethernet interfaces, the Xilinx Virtex II pro FPGA (though now is outdated), 2 power PC cores, and uses the standard form factor PCI. This DNS server implementation uses the NetFPGA 1G card for use in its design and below is a diagram describing the major components of the NetFPGA 1G.

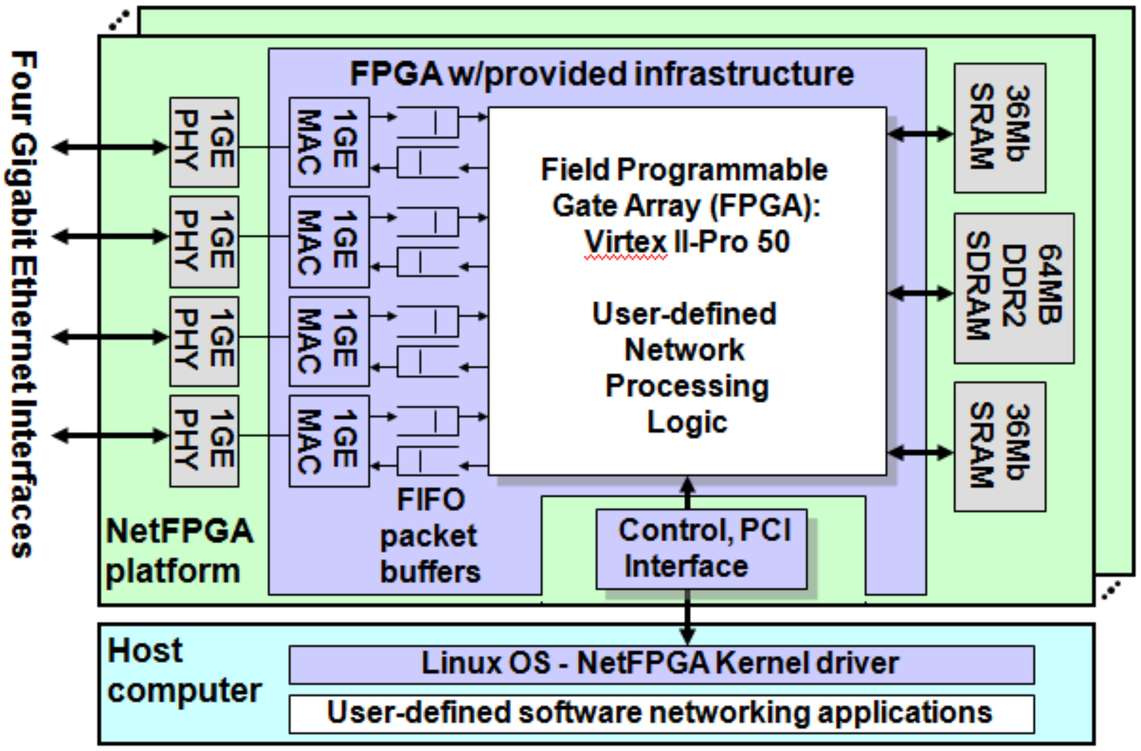

Four Gigabit Ethernet Interfaces
1GE PHY
1GE PHY
1GE PHY
1GE PHY
FPGA w/provided infrastructure
1GE MAC
1GE MAC
1GE MAC
1GE MAC
Field Programmable Gate Array (FPGA): Virtex II-Pro 50
User-defined Network Processing Logic
36Mb SRAM
64MB DDR2 SDRAM
36Mb SRAM
NetFPGA platform
FIFO packet buffers
Control, PCI Interface
Host computer
Linux OS - NetFPGA Kernel driver
User-defined software networking applications

Figure 13 Block Diagram NetFPGA 1G

The NetFPGA project contains an API or suite of libraries, modules, and reference designs available for use of the developer. The API provides an easy way for the developer to program the device. The API has a variety of components: C, Java, makefiles, Perl, Python, scripts, Verilog, and XML. Using the API, a developer could create a new project without having to write any Verilog modules. The developer uses the PCI bus to load new design onto the NetFPGA through a bitfile created through the NetFPGA API. Also, a reference pipeline describes the data path to where modules could be added if necessary.

3.1.1 Reference Pipeline

The reference pipeline defines the data movement through the modules on the NetFPGA. The MAC RX and TX queues correspond to the ethernet interfaces on the NetFPGA device. The CPU RX and TX queues correspond to the packets that come from the software on the corresponding interface. The Input Arbiter is a module that determines which RX queue to pull a packet. The Output Port Lookup is a module that determines what output port a packet will be transmitted. The Output Queues is a module that places the packet in an output queue corresponding with the output port until the corresponding TX port can send the packet.

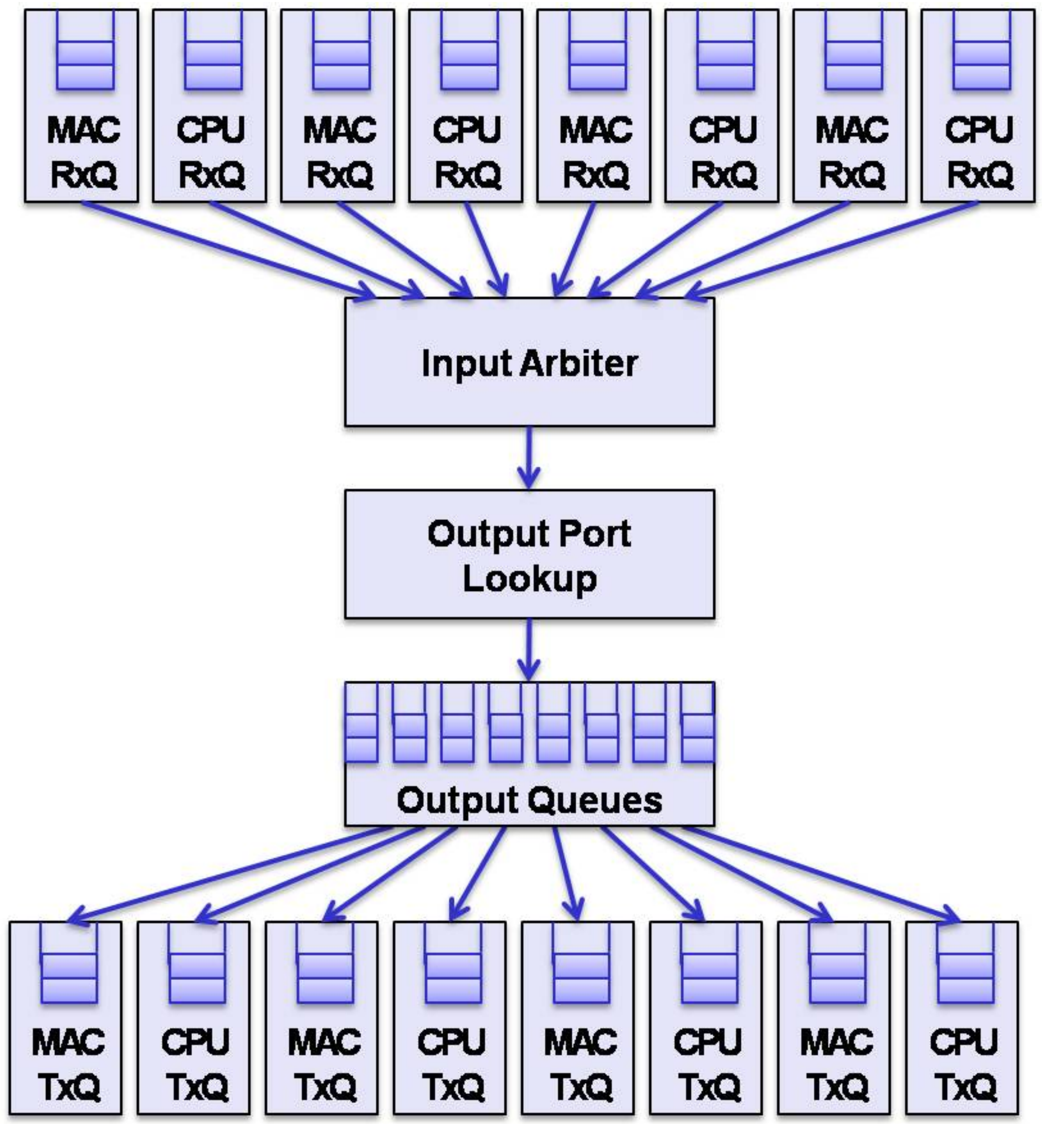


Figure 14 Reference Diagram

The Input Arbiter, Output Port Lookup, and the Output Queues are part of the User Data Path. The Input Arbiter begins the User Data Path and the Output Queues is the end. New Modules are added in after the Input Arbiter and before the Output Queues. However, they may be added either before or after the Output Port Lookup.

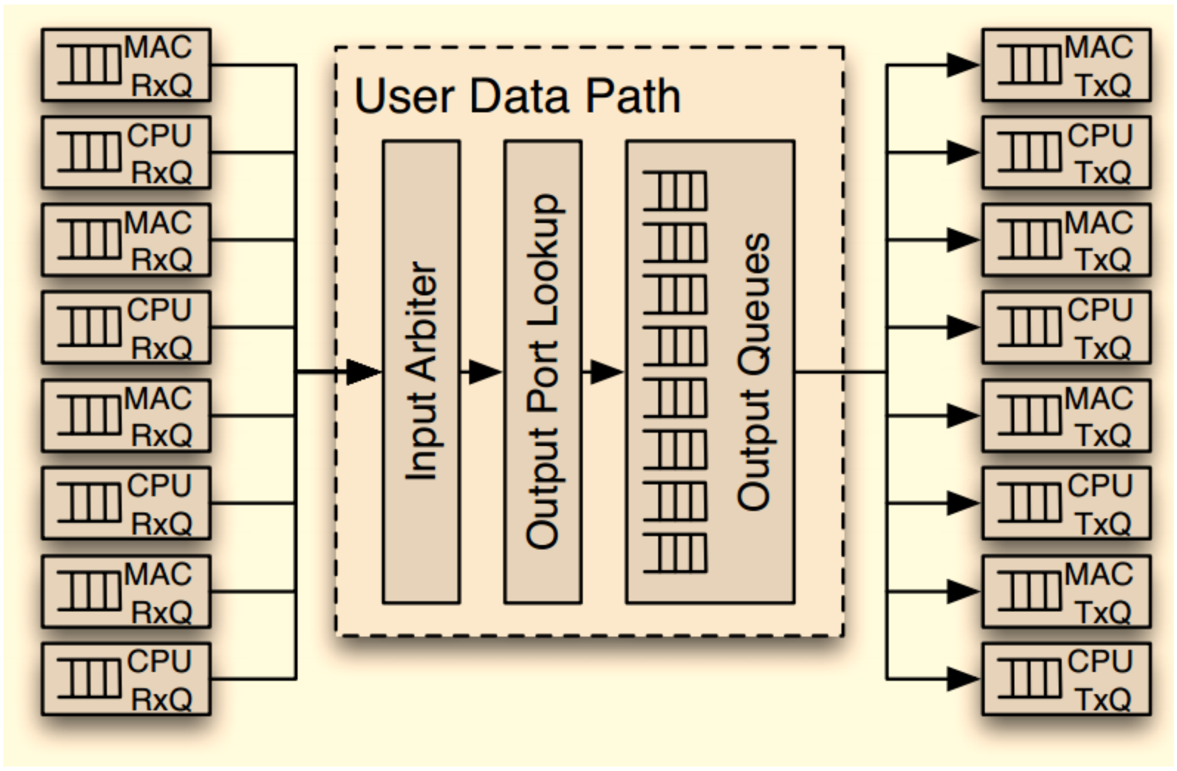


Figure 15 Reference Diagram with User Data Path

The User Data Path defines how modules pass data to the next module. The User Data Path has a 64-bit data bus, an 8-bit control bus, a 1-bit write line, and a 1-bit ready line. The write line alerts the next module it has a packet to pass on to it. The ready line alerts the previous module that it is available to receive a packet. The data bus is where the packet is sent (as well as some module header information). The control bus determines whether the information on the data bus is a module header, packet data, or the end of a packet.

The module headers are determined by non-zero ctrl bus signals and are sent first. Afterwards, the ctrl bus sets to zero determining the data bus has packet data. The packet end is determined by setting a 1 in the bit corresponding with the last valid byte, calculated by left shift 1 ((8-number of valid bytes) modulus 8), and the bytes are stored in the most significant position in the data bus.

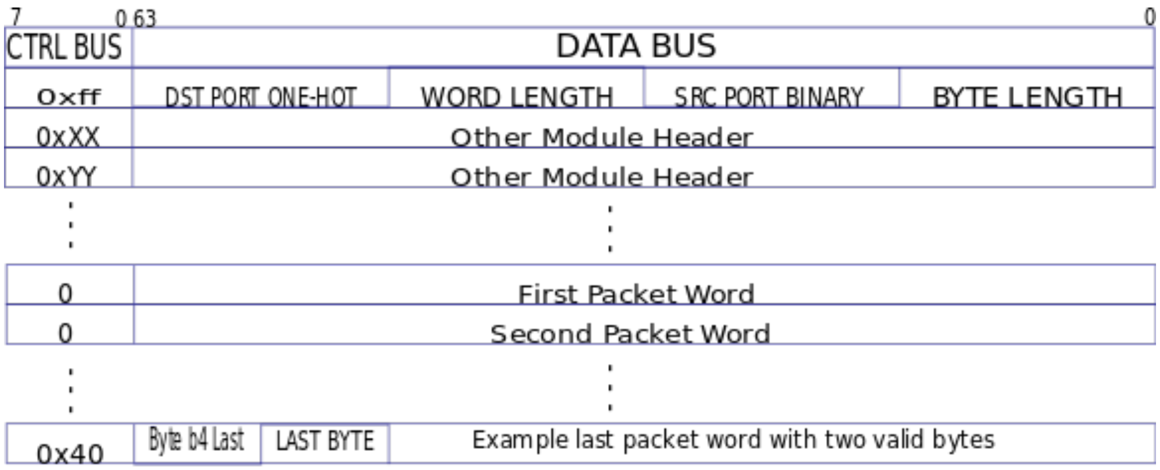


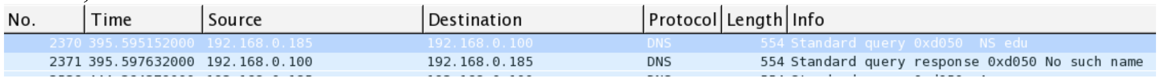
Figure 16 Packet Format in Hardware Pipeline

3.2 Recent Additions to the NetFPGA family

In 2009, the NetFPGA 10G was released with larger memory, faster 10Gbps ethernet interfaces, an updated FPGA, larger and faster SRAM, larger and faster DDR2 RAM, and uses PCI express gen 1. Also, more recent additions to the NetFPGA family include the 1G CML and the SUME. The 1G CML appears to be an update to the NetFPGA 1G with 4x1Gbps ethernet interfaces, with faster SRAM, faster and larger DRAM (DDR 3), PCI express gen2, and a storage slot for a SD card. The SUME appears to be an update to the NetFPGA 10G with faster SRAM, faster and grossly larger DRAM, PCI express gen 3, and a micro SD storage slot.

4 The Project

The original goal of this project was to create a hardware-based DNS server using the NetFPGA and compare its performance to a software version running on a host computer. Due to complications with the DNS lookup Trie structure in the hardware design a comparison could not occur. However, the DNS server was put onto the NetFPGA and was able to respond to queries. Although, those responses were not very useful, a hardware based DNS server was created.



```
No.   Time           Source           Destination      Protocol Length Info
 2370 395.595152000  192.168.0.185    192.168.0.100    DNS         554 Standard query 0xd050  NS edu
 2371 395.597632000  192.168.0.100    192.168.0.185    DNS         554 Standard query response 0xd050 No such name
```

Figure 17 Hardware DNS server response

4.1 DNS Lookup Structure

Typically a DNS server's implementation of the lookup structure is a hashed structure, although there is no stipulation on how lookups are performed on a DNS server. The hash structure is used for its performance in accomplishing lookups given a good hash function. After a discussion on hardware lookup structure, and being directed to various papers on the use of Trie structures in hardware NetFPGA based IP routers, using a Trie as the DNS lookup structure seemed appropriate. Since IP addresses and domain names share a hierarchical structure, the prefix Trie structure looked like a great candidate. There is a difference between IP lookup and DNS; where IP has a simple port lookup, DNS has a more complicated consideration of the resource records.

There has been great research done with optimizing and designing efficient Trie structures for use in IP routing on the NetFPGA. Optimized Linear Pipeline (OLP) Circular Adaptive Monotonic Pipeline (CAMP) and the ring pipeline are examples of some optimized Trie structures designed for FPGA's use in IP lookups. These optimizations are based upon pipelining the lookup by breaking apart the Trie structure into stages of sub-tries. These optimizations were considered; however, a simple adapted Trie structure seemed appropriate, as further optimizations could be explored at a later point. This adapted Trie structure would be a basic prefix Trie where the nodes would also contain a pointer to a structure that had pointers for each resource record type. Depending on the type, the resource record would either be a stand-alone record or a linked list of that resource record type, since some domain names may have multiple records of certain types. The stand-alone records are: SOA, CNAME, and PTR. The records that a domain name may have multiples, or linked list records, are: A, NS, MX and AAA.

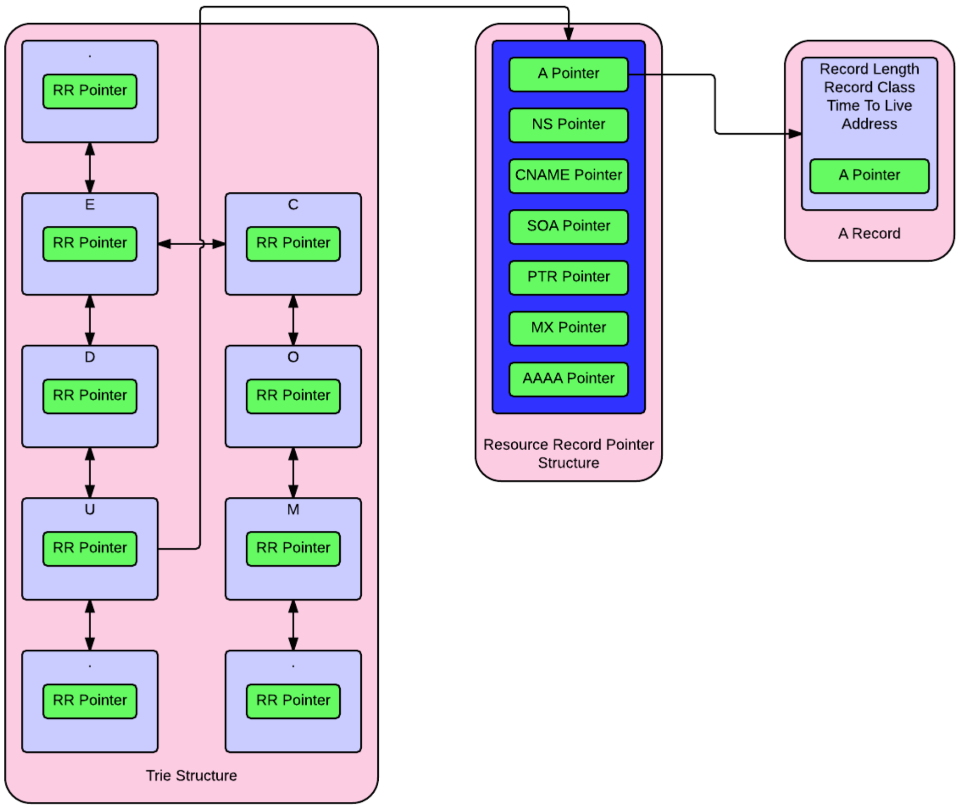


Figure 18 Trie Diagram

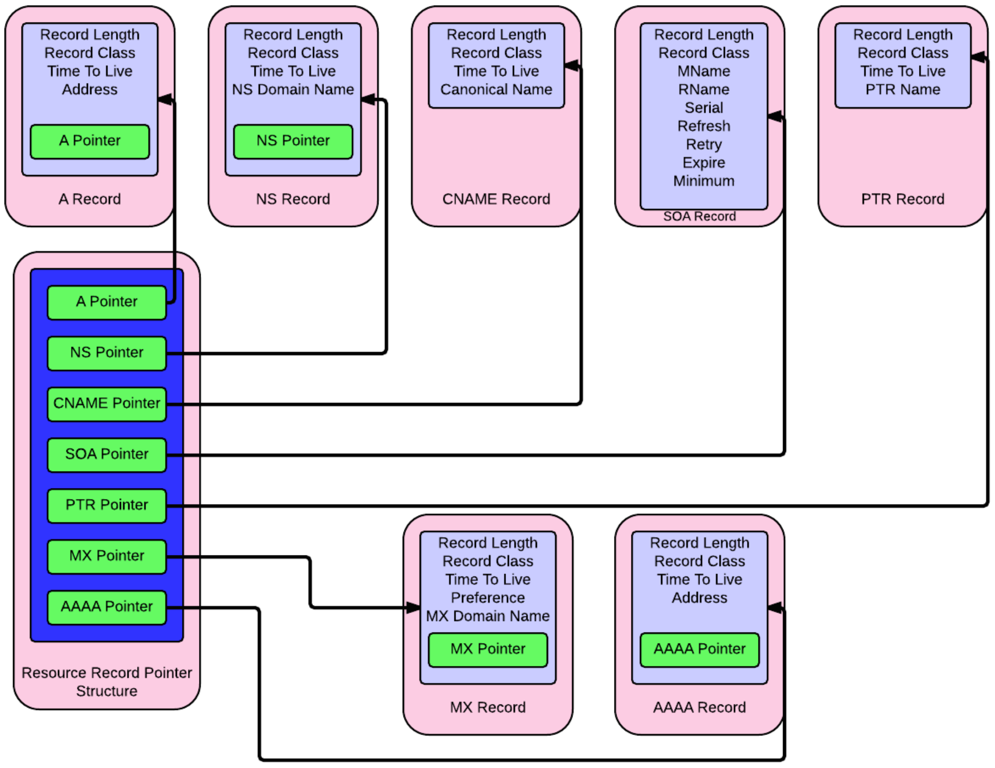


Figure 19 Record Structure Diagram

4.2 Software Design

A software version implemented on a host running Fedora 19 using this Trie structure was created first, for comparison, as a reference, and for better understanding of the DNS server. This helped in working out the functionality of the Trie structure. The software version helped in figuring out how to parse the Master File and load the records into the Trie structure. Also, this version helped in understanding procedures in query restarts, where the query would have to restart the query with the domain name given in the resource record (in CNAME and NS queries where the query type was not of that type). Several adaptations of this software version were used in attempts to load the Trie structure on the NetFPGA.

This version utilized the network socket API for the network communication. The software version was not a great reference for the lower level network protocols, since the API makes it easy to implement UDP and made it unnecessary to code the lower-layer protocols in the network stack. However, the Master File still needed to be parsed, the Trie structure still required loading (though with pointers and the creation of structures was easier than working directly in Verilog), using the Trie structure to lookup resource records, and the DNS packets still needed to be parsed and acted upon.

The parsing of the DNS packets greatly helped in understanding what needed to be done in the hardware. Pulling apart the header and extracting the query information made it easy to design a hardware version. Much of that work could be directly translated into Verilog, while some would prove to be simpler in Verilog.

The parsing of the Master File and loading its contents into the Trie structure aided in understanding the resource records and how the structure would be represented in memory. The parsing and loading sections was used in a couple attempts of loading the structure in hardware. However, the abstraction and the memory management that the operating system provides made it easier to perform the creation of the Trie structure.

The lookup is simpler to implement in software than in hardware using a low level language like Verilog. The use of pointers and structure abstracts much of the lower level functions that would be required in a lower level language like Verilog. However, the general design of the structure is the same.

4.3 Hardware Design

The search for a way to implement a hardware based DNS server began with designing the memory structure in which the DNS server performs the lookup to find the corresponding resource record. After deciding upon the Trie structure, a rough design of the architecture began to take shape. Early designs incorporated the parsing of the lower network level protocols specific only to DNS packets and the lookup. However, parsing the zone file proved difficult, as Verilog does not perform file I/O in synthesizable code.

The original hardware design, Figure 20 below, did not take into account the lower layers in the network stack, the data-link, or the network layer. This original hardware design was based off the software host design, which utilized the Network Socket API, where only the transport and application required attention. The basis was that there would be built in mechanism to help process the lower layers in the network model. However, it did not take long to realize that the lower layers would have to be addressed.

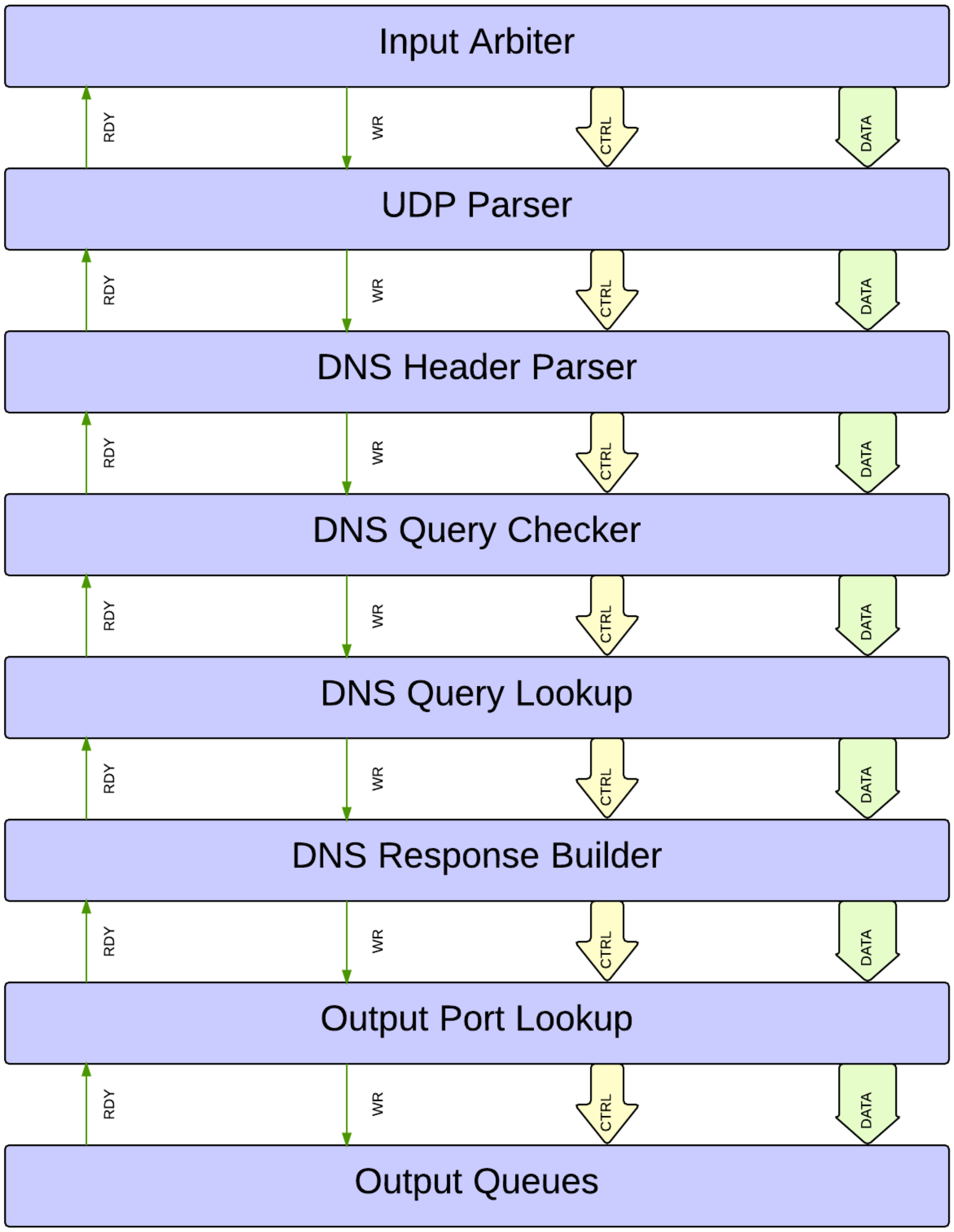


Figure 20 Original DNS Hardware Design NetFPGA User Data Path

The redesign took into account the lower layers of the network stack. The data-link layer, which corresponds to the ethernet frame particularly the header, required parsing for decisions to be made upon the data discerned from this layer. The network layer, which corresponds to the IP header/packet, needed to be parsed as well. Only after traversing through those layers could the layers coded in the software version be addressed. This redesign was the basis for the initial code attempts with the NetThreads compiler.

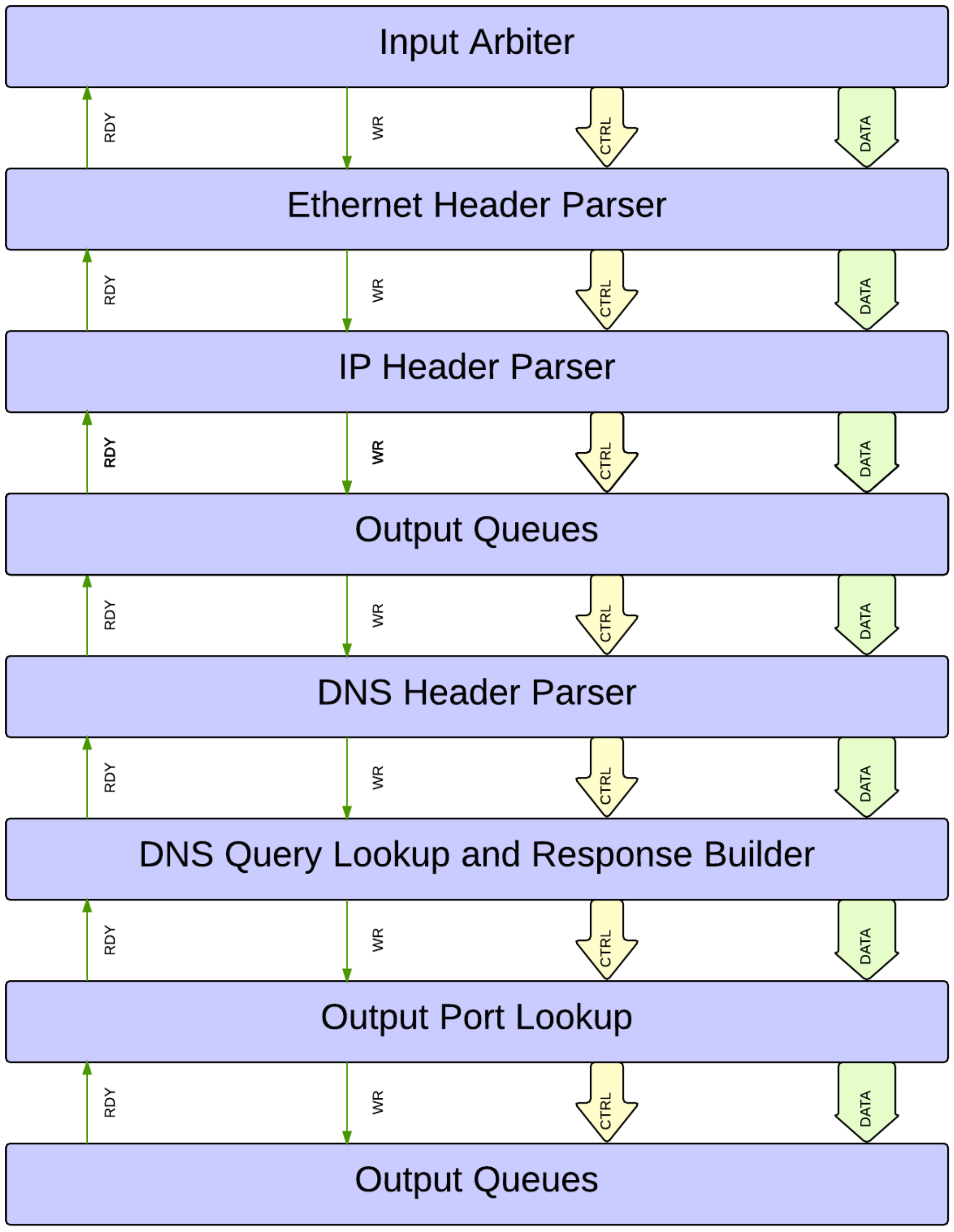


Figure 21 DNS Hardware Design NetFPGA User Data Path Revision

During the use of the NetThreads compiler, it became apparent that other protocols needed to be utilized. The ARP and ICMP Echo replies were used to establish and help discern issues with connectivity to the device. The design below (Figure 22) depicts the basic process flow for the final design of the DNS server using the NetThreads compiler.

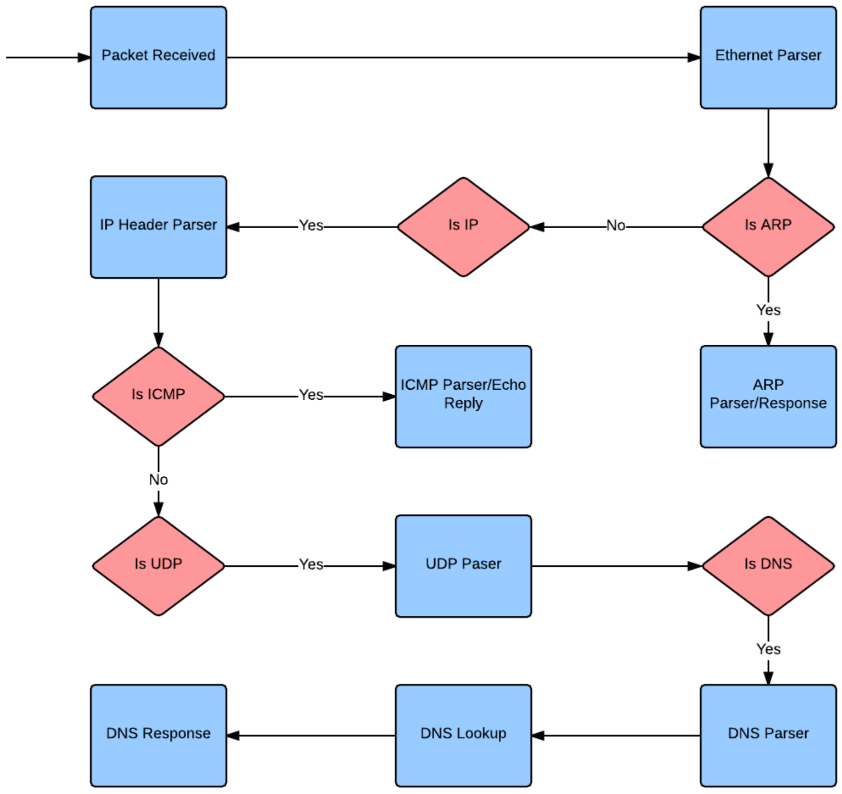


Figure 22 NetThreads DNS Design Workflow

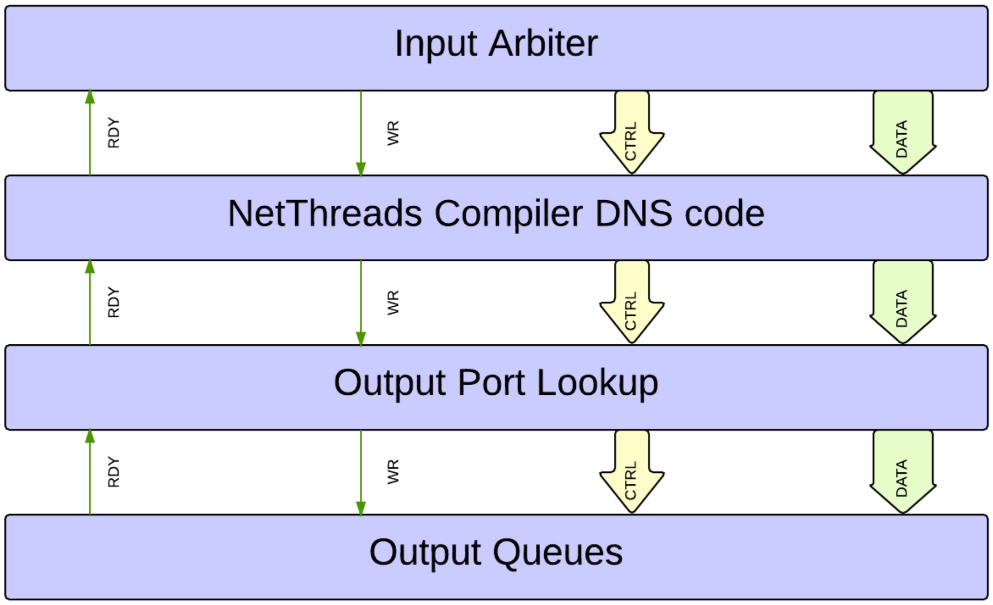


Figure 23 NetThreads DNS Design within the NetFPGA User Data Path

Due to complications with the lookup structure with the NetThreads compiler, a design was imagined for Verilog modules. This design was based upon the NetThreads process flow. This design could be used for the basis of a NetFPGA design using only the Verilog modules instead of the NetThreads compiler.

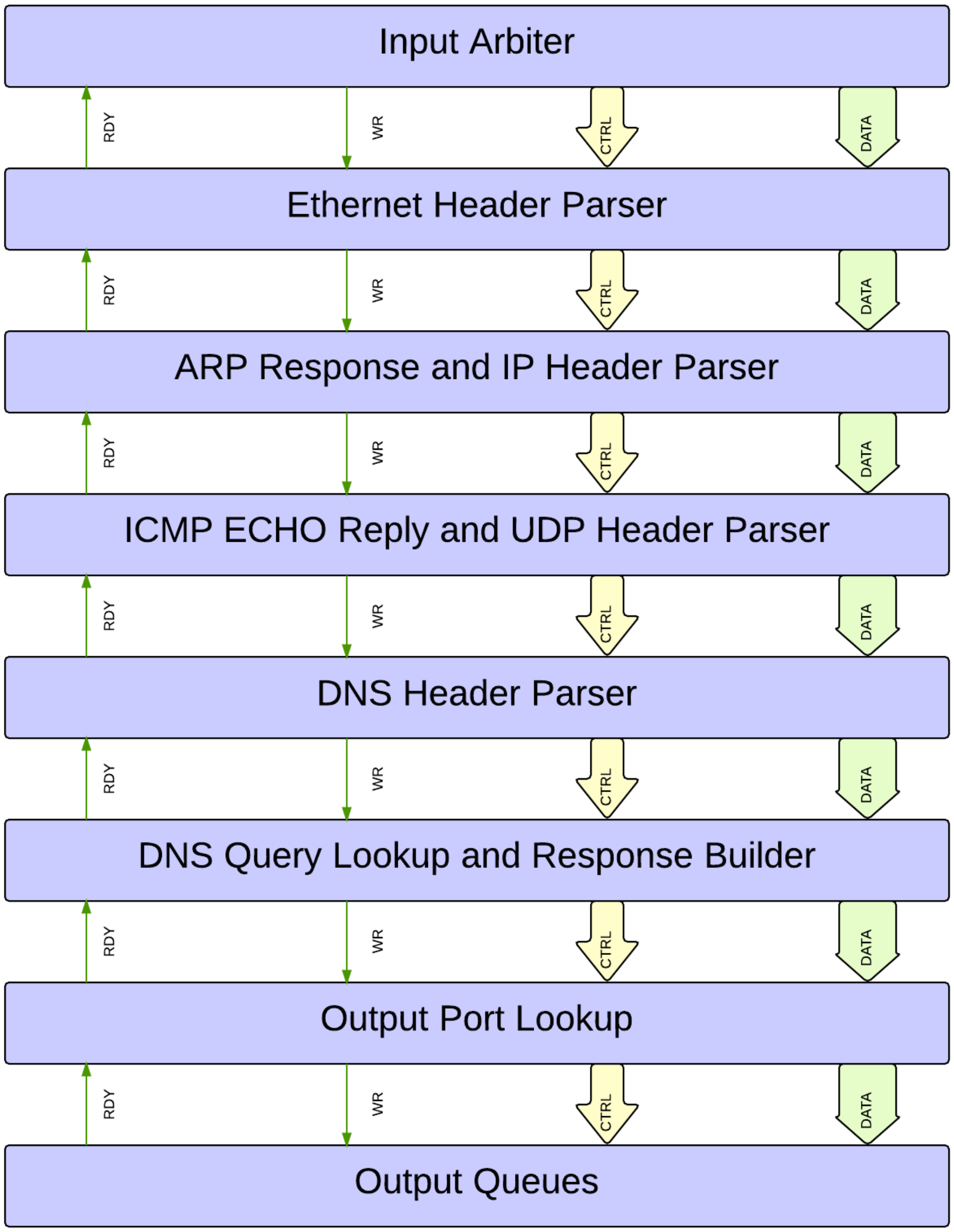

Input Arbiter
RDY
WR
CTRL
DATA
Ethernet Header Parser
ARP Response and IP Header Parser
ICMP ECHO Reply and UDP Header Parser
DNS Header Parser
DNS Query Lookup and Response Builder
Output Port Lookup
Output Queues

Figure 24 DNS Hardware Final Design NetFPGA User Data Path

4.3.1 NetThreads

In searching for a solution to performing file I/O, the discovery of a compiler NetFPGA project occurred. The NetThreads project along with NetThreads-Re and NetTM are NetFPGA projects that allow a developer to program in C, providing some abstraction to the lower level Verilog language. Upon suggestion of the compilers' creator, the NetThreads project was chosen, as it was the most stable of the three compiler projects. The NetThreads project utilizes the programmable logic as a soft processor, and in the Reference Pipeline, it exists between the Input Arbiter and the Output Queues. However, this did not solve the problem since I/O operations are not permitted. This did open the opportunity to increase the translation of the already written software DNS server.

Translating the software version had its difficulties, though was more straightforward and easier than translating into Verilog. There were some adjustments that had to be made and some operations required rework, as the lightweight version of C used by NetThreads does not have extensive libraries. Creating the parsing logic of the lower level network protocols needed to be done. Also, working through compiler errors and adjusting to the API required some time and some code reading. Though when the project compiled and the code was loaded onto the NetFPGA, it did not work or respond.

The project has a debug feature, though that feature would also not work properly. The debug feature would compile a version of the code that would run on the host computer but it would not compile the software version. The example programs were revisited and the basic ‘hello world’ equivalent was the only successful initial recreation. The example ping program, or ICMP Echo reply program would not respond. Debugging the ping program’s lack of a response was a tedious and fairly long process. The initial thought was that the version of the NetFPGA development package was incompatible from the version that NetThreads was originally developed upon. That was not the case, as the environment used in developing NetThreads was closely created and still the code was non-responsive.

It took long discussion with networking experts, more sophisticated packet tools, a managed layer 3 switch, and some research into lower level network protocols to debug why a simple ICMP Echo request would not work. The managed layer 3 switch showed that the network did not know the device existed. Even after configuring the switch and the router with the device’s IP and MAC address, the network simply would not recognize the device. Through research and use of Wireshark, the culprit was discovered to be a lack of responses and requests from ARP.

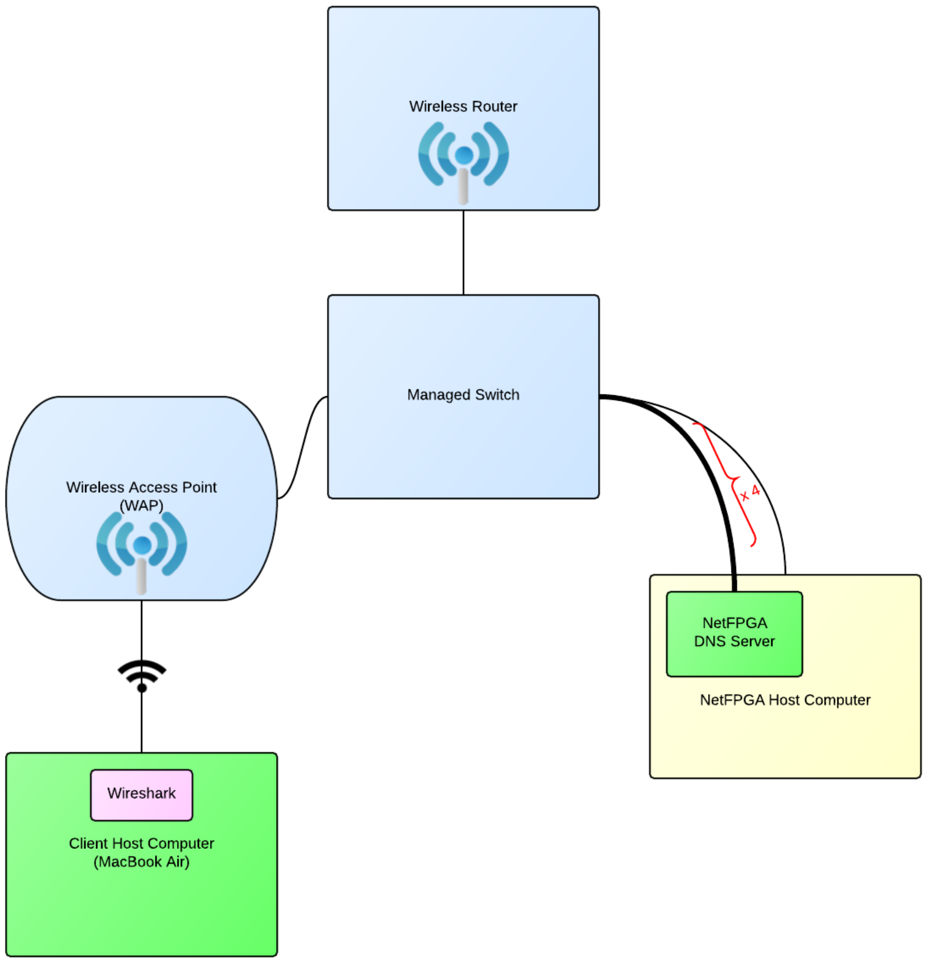


Figure 25 Physical setup showing where Wireshark ran for debugging

The creation of a simple ARP program and a new ping program were the first steps in debugging network issues with the DNS server. Afterwards came a tedious process of fixing issues by reworking the process flow of the DNS server. The following sections briefly describe the various network layers and the issues encountered in the layers.

4.3.2 Brief overview of the 7-Layer Open System Interconnection (OSI) Model

The 7-layer OSI model defines the various layers of the network stack. The layers allow for the complex system to be broken apart into manageable means for communication. Within each layer are protocols, which govern the layer's purpose and the interactions for that layer. The lower layers deal with the logistics of sending data physically and across large areas. The upper layers deal with the logistics of the communication of the end hosts.

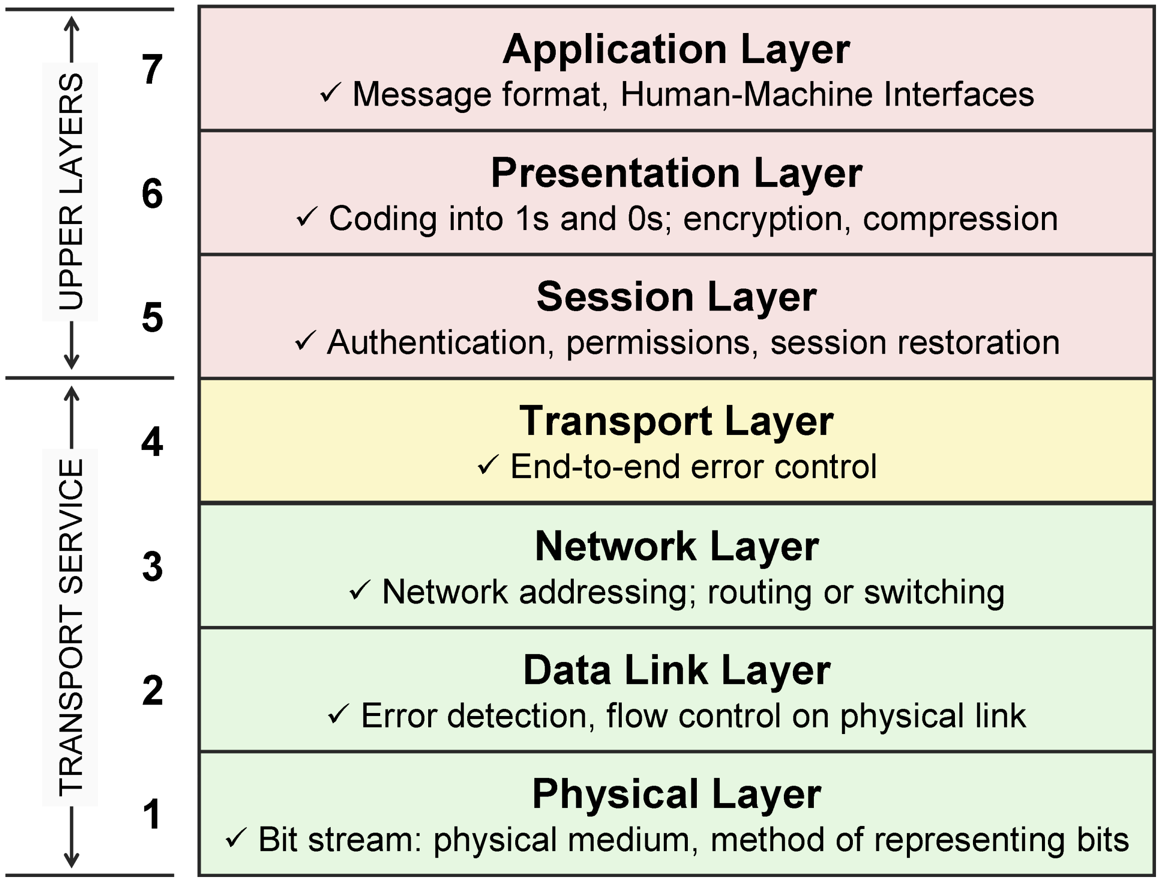

UPPER LAYERS
TRANSPORT SERVICE
7
6
5
4
3
2
1
Application Layer
✓ Message format, Human-Machine Interfaces
Presentation Layer
✓ Coding into 1s and 0s; encryption, compression
Session Layer
✓ Authentication, permissions, session restoration
Transport Layer
✓ End-to-end error control
Network Layer
✓ Network addressing; routing or switching
Data Link Layer
✓ Error detection, flow control on physical link
Physical Layer
✓ Bit stream: physical medium, method of representing bits

Figure 26 Open System Interconnection (OSI) 7-Layer Model

4.3.3 Ethernet II Frames: data-link layer (layer 2)

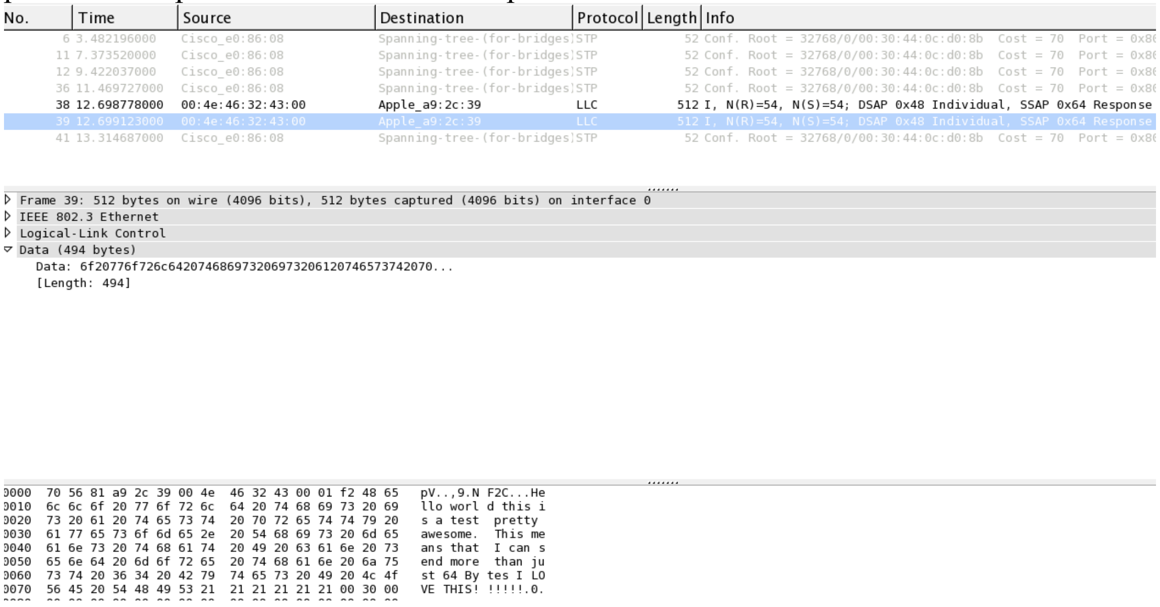
Ethernet II frames exist at the data-link layer and provide transit of packets through ethernet interfaces over ethernet cables. Various grades of ethernet interfaces and cable deal with the payload thresholds being carried across the network. The NetFPGA 1G contains 4 1-gigibit per seconds interfaces. Ethernet frames encapsulate the packet and the higher layer protocols.

The ethernet II header states the **source address**, **destination address**, and the ethernet type. The **source address** is the MAC address, or the machine address, of the sender and the **destination address** is the receiver of the intended packet. The ethernet **type** determines how to interpret the encapsulated data that corresponds to the higher-level protocols; in our case we are only concerned with ARP (0x0806) and IP (0x0800). The **frame check sequence** is filled using the Cyclic Redundancy Check (CRC) algorithm.

Preamble	Destination MAC address	Source MAC address	Type/Length	User Data	Frame Check Sequence (FCS)
8	6	6	2	46 - 1500	4

Figure 27 Ethernet II frame

A simple ‘hello world’ program was created and established that the device was working properly. Also, it established that the NetThreads project was able to send packets. A capture of the ‘hello world’ packet sent to the host is shown below.



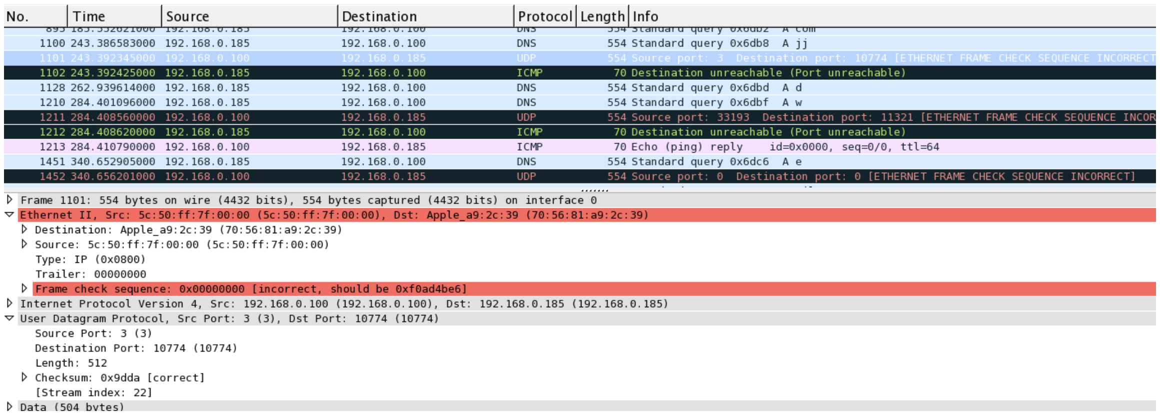
```
No.   Time          Source             Destination                 Protocol Length Info
    6 3.482196000   Cisco_e0:86:08     Spanning-tree-(for-bridges)STP      52 Conf. Root = 32768/0/00:30:44:0c:d0:8b  Cost = 70  Port = 0x8
   11 7.373520000   Cisco_e0:86:08     Spanning-tree-(for-bridges)STP      52 Conf. Root = 32768/0/00:30:44:0c:d0:8b  Cost = 70  Port = 0x8
   12 9.422037000   Cisco_e0:86:08     Spanning-tree-(for-bridges)STP      52 Conf. Root = 32768/0/00:30:44:0c:d0:8b  Cost = 70  Port = 0x8
   36 11.469727000  Cisco_e0:86:08     Spanning-tree-(for-bridges)STP      52 Conf. Root = 32768/0/00:30:44:0c:d0:8b  Cost = 70  Port = 0x8
   38 12.698778000  00:4e:46:32:43:00  Apple_a9:2c:39              LLC    512 I, N(R)=54, N(S)=54; DSAP 0x48 Individual, SSAP 0x64 Response
   39 12.699123000  00:4e:46:32:43:00  Apple_a9:2c:39              LLC    512 I, N(R)=54, N(S)=54; DSAP 0x48 Individual, SSAP 0x64 Response
   41 13.314687000  Cisco_e0:86:08     Spanning-tree-(for-bridges)STP      52 Conf. Root = 32768/0/00:30:44:0c:d0:8b  Cost = 70  Port = 0x8

▷ Frame 39: 512 bytes on wire (4096 bits), 512 bytes captured (4096 bits) on interface 0
▷ IEEE 802.3 Ethernet
▷ Logical-Link Control
▽ Data (494 bytes)
    Data: 6f20776f726c642074686973206973206120746573742070...
    [Length: 494]

0000  70 56 81 a9 2c 39 00 4e  46 32 43 00 01 f2 48 65   pV..,9.N F2C...He
0010  6c 6c 6f 20 77 6f 72 6c  64 20 74 68 69 73 20 69   llo worl d this i
0020  73 20 61 20 74 65 73 74  20 70 72 65 74 74 79 20   s a test  pretty
0030  61 77 65 73 6f 6d 65 2e  20 54 68 69 73 20 6d 65   awesome.  This me
0040  61 6e 73 20 74 68 61 74  20 49 20 63 61 6e 20 73   ans that  I can s
0050  65 6e 64 20 6d 6f 72 65  20 74 68 61 6e 20 6a 75   end more  than ju
0060  73 74 20 36 34 20 42 79  74 65 73 20 49 20 4c 4f   st 64 By tes I LO
0070  56 45 20 54 48 49 53 21  21 21 21 21 21 00 30 00   VE THIS! !!!!!.0.
```

Figure 28 NetThreads’ Hello World

During the early stages of debugging, there were errors in the ethernet header and the frame check sequence. The error was the result of an issue with the offset. The offset was corrected and the error was resolved.

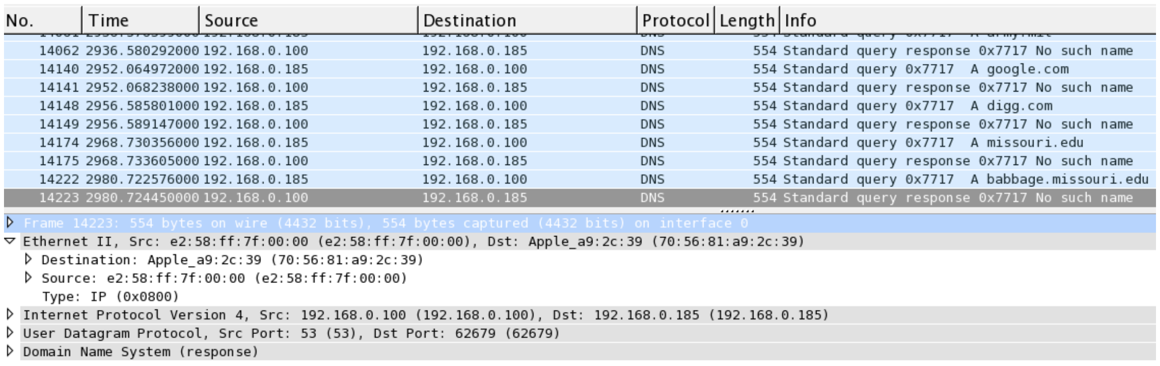


```
No.   Time           Source         Destination    Protocol Length Info
 1100 243.386583000  192.168.0.185  192.168.0.100  DNS       554 Standard query 0x6db8  A jj
 1101 243.392345000  192.168.0.100  192.168.0.185  UDP       554 Source port: 3  Destination port: 10774 [ETHERNET FRAME CHECK SEQUENCE INCORRECT
 1102 243.392425000  192.168.0.185  192.168.0.100  ICMP       70 Destination unreachable (Port unreachable)
 1128 262.939614000  192.168.0.185  192.168.0.100  DNS       554 Standard query 0x6dbd  A d
 1210 284.401096000  192.168.0.185  192.168.0.100  DNS       554 Standard query 0x6dbf  A w
 1211 284.408560000  192.168.0.100  192.168.0.185  UDP       554 Source port: 33193  Destination port: 11321 [ETHERNET FRAME CHECK SEQUENCE INCOR
 1212 284.408620000  192.168.0.185  192.168.0.100  ICMP       70 Destination unreachable (Port unreachable)
 1213 284.410790000  192.168.0.100  192.168.0.185  ICMP       70 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64
 1451 340.652905000  192.168.0.185  192.168.0.100  DNS       554 Standard query 0x6dc6  A e
 1452 340.656201000  192.168.0.100  192.168.0.185  UDP       554 Source port: 0  Destination port: 0 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]

▷ Frame 1101: 554 bytes on wire (4432 bits), 554 bytes captured (4432 bits) on interface 0
▽ Ethernet II, Src: 5c:50:ff:7f:00:00 (5c:50:ff:7f:00:00), Dst: Apple_a9:2c:39 (70:56:81:a9:2c:39)
  ▷ Destination: Apple_a9:2c:39 (70:56:81:a9:2c:39)
  ▷ Source: 5c:50:ff:7f:00:00 (5c:50:ff:7f:00:00)
    Type: IP (0x0800)
    Trailer: 00000000
  ▷ Frame check sequence: 0x00000000 [incorrect, should be 0xf0ad4be6]
▷ Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.185 (192.168.0.185)
▽ User Datagram Protocol, Src Port: 3 (3), Dst Port: 10774 (10774)
    Source Port: 3 (3)
    Destination Port: 10774 (10774)
    Length: 512
  ▷ Checksum: 0x9dda [correct]
    [Stream index: 22]
▷ Data (504 bytes)
```

Figure 29 NetFPGA DNS response bad Ethernet header good UDP checksum

Late in the project the ethernet source address was found to be incorrect. This was due to how functions pass variables. It appeared there was a limit to the variables that could be passed from function to function. When passing function to function, the variable was corrupted and could not be used. Once this limit was found, the solution was simple and the ethernet source address was fixed.



```
No.   Time              Source          Destination     Protocol Length Info
14062 2936.580292000 192.168.0.100   192.168.0.185   DNS      554 Standard query response 0x7717 No such name
14140 2952.064972000 192.168.0.185   192.168.0.100   DNS      554 Standard query 0x7717  A google.com
14141 2952.068238000 192.168.0.100   192.168.0.185   DNS      554 Standard query response 0x7717 No such name
14148 2956.585801000 192.168.0.185   192.168.0.100   DNS      554 Standard query 0x7717  A digg.com
14149 2956.589147000 192.168.0.100   192.168.0.185   DNS      554 Standard query response 0x7717 No such name
14174 2968.730356000 192.168.0.185   192.168.0.100   DNS      554 Standard query 0x7717  A missouri.edu
14175 2968.733605000 192.168.0.100   192.168.0.185   DNS      554 Standard query response 0x7717 No such name
14222 2980.722576000 192.168.0.185   192.168.0.100   DNS      554 Standard query 0x7717  A babbage.missouri.edu
14223 2980.724450000 192.168.0.100   192.168.0.185   DNS      554 Standard query response 0x7717 No such name

▷ Frame 14223: 554 bytes on wire (4432 bits), 554 bytes captured (4432 bits) on interface 0
▽ Ethernet II, Src: e2:58:ff:7f:00:00 (e2:58:ff:7f:00:00), Dst: Apple_a9:2c:39 (70:56:81:a9:2c:39)
  ▷ Destination: Apple_a9:2c:39 (70:56:81:a9:2c:39)
  ▷ Source: e2:58:ff:7f:00:00 (e2:58:ff:7f:00:00)
    Type: IP (0x0800)
▷ Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.185 (192.168.0.185)
▷ User Datagram Protocol, Src Port: 53 (53), Dst Port: 62679 (62679)
▷ Domain Name System (response)
```

Figure 30 NetFPGA DNS response appears good, but Ethernet header contains incorrect source address

4.3.4 ARP: Address Resolution Protocol: data-link layer/network layer protocol

Address resolution protocol (ARP) is a part of both data-link and network layer protocol. ARP provides a method for an address, IPv4 for this project, to be translated into a MAC address. In other words, it provides a means for a network layer address to resolve into a data-link layer address so that a packet can be formed and sent. ARP is a means for a device/host to notify the network of its presence. Routers, switches, and hosts maintain an ARP cache, a table of IP addresses to MAC addresses, which enable creation and forwarding of packets. ARP is a discovery mechanism for devices/hosts for the network.

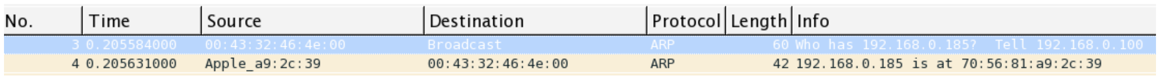
ARP requests and ARP replies also act as a means for how the network discovers devices. ARP requests broadcast a message across the network asking who has this IP address. An ARP reply is a response to that request notifying the network that this MAC address has that IP address. Typically, the ARP cache is updated with that IP address to MAC address. Periodically, the ARP requests are resent in order to refresh the ARP cache with current configurations.

The ARP packet for ethernet frames is shown in the image below. The ARP header is defined in green or the top 64-bits (8 bytes). The **HRD** is the format of the hardware address; in this case 1 to indicate it is from an ethernet frame 10/100Mbps ethernet. The PRO is the format of the protocol address; in this case 0x0800 to indicate that the protocol is an IPv4 address. The **HLN** and **PLN** are the lengths of the hardware and protocol address in bytes. In this case the **HLN** is 6 and **PLN** is 4. The **OP** is the operational code and for the DNS server the values are 1 to signify a request (sent during startup of the device) and 2 to signify a reply. The **sender hardware address** is the 48-bit MAC address of the sender in both requests and replies. The **target hardware address** is the 48-bit ethernet address of the target in both requests and replies, and in requests it is set to the broadcast ethernet address (FF:FF:FF:FF:FF:FF). The **sender protocol address** is the 32-bit IPv4 address of the sender in both requests and replies. The **target protocol address** is the 32-bit IPv4 address of the target in both requests and replies.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HRD (Format of Hardware Address)															
PRO (Format of Protocol Address)															
HLN (Length of Hardware Address)								PLN (Length of Protocol Address)							
OP (Operational Code)															
Sender Hardware Address															
Sender Hardware Address continued															
Sender Hardware Address continued															
Sender Protocol Address															
Sender Protocol Address continued															
Target Hardware Address															
Target Hardware Address continued															
Target Hardware Address continued															
Target Protocol Address															
Target Protocol Address continued															

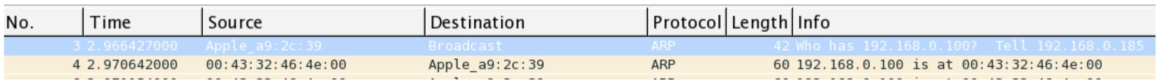
Figure 31 Format of Ethernet ARP Packet

During the course of the project, the discovery of the NetFPGA device was not occurring. The debugging process included several talks with network experts and use of tools and special devices. It took several months to discover the issue and required the use of Wireshark and acquisition of a managed switch (layer 3 Cisco switch) in order to discover that the NetFPGA device was not sending any packets. The network simply did not know the NetFPGA existed. The resolution was that the NetFPGA device had to send ARP requests out to the network during startup. In order to make certain that the necessary components knew of the NetFPGA's existence, the NetFPGA sent out ARP requests to the switch, the router, and the client host.



```
No. Time         Source             Destination        Protocol Length Info
  3 0.205584000  00:43:32:46:4e:00  Broadcast          ARP          60 Who has 192.168.0.185?  Tell 192.168.0.100
  4 0.205631000  Apple_a9:2c:39     00:43:32:46:4e:00  ARP          42 192.168.0.185 is at 70:56:81:a9:2c:39
```

Figure 32 ARP Request success from NetFPGA

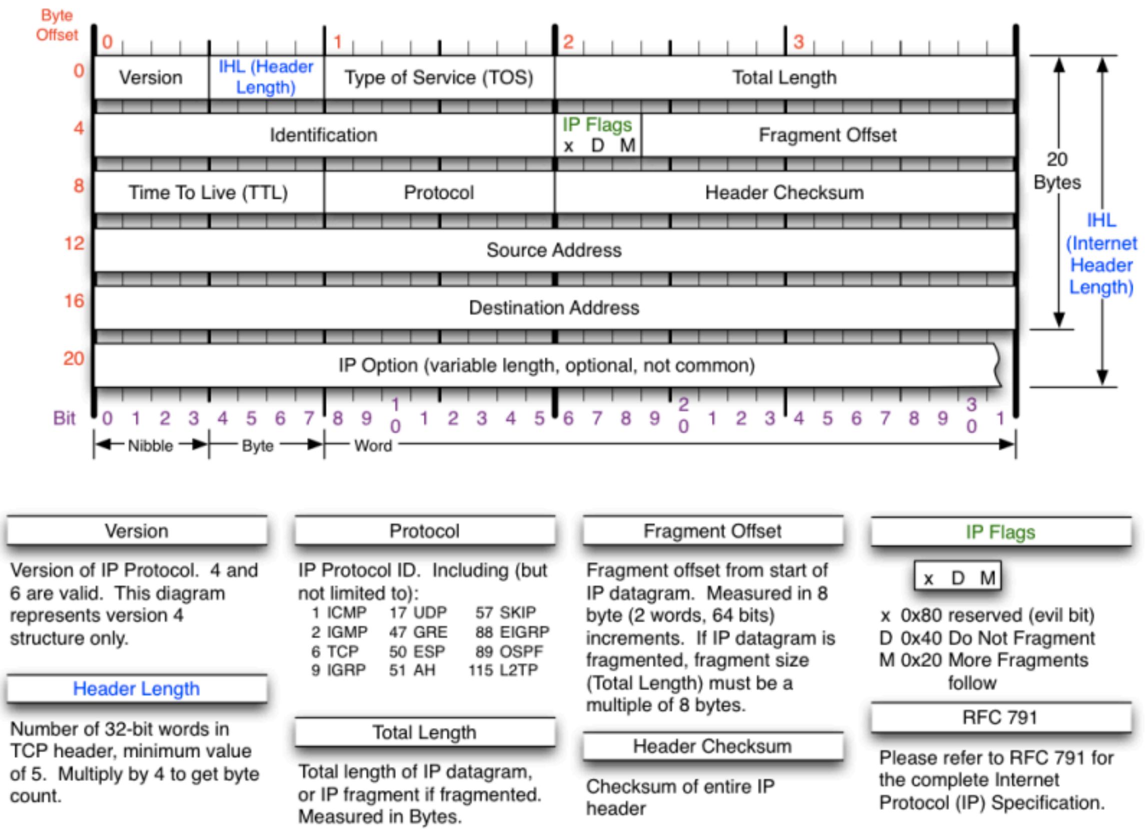


```
No. Time         Source             Destination        Protocol Length Info
  3 2.966427000  Apple_a9:2c:39     Broadcast          ARP          42 Who has 192.168.0.100?  Tell 192.168.0.185
  4 2.970642000  00:43:32:46:4e:00  Apple_a9:2c:39     ARP          60 192.168.0.100 is at 00:43:32:46:4e:00
```

Figure 33 ARP Reply success from NetFPGA

4.3.5 IP: Internet Protocol: network layer (layer 3)

Internet Protocol, IP, is a network layer protocol in which datagrams are sent over a network. The **version** and **IHL** denote the version and the sizes of the address fields, in this case the IPv4 or 32-bit addresses. The **total length** denotes the size of the IP packet including the header. The **TTL** field is the time to live of the packet. The **protocol** denotes the encapsulated protocol of the higher network layer, for the purposes in the design either ICMP or UDP. The header **checksum** is a 16-bit one's complement of the IP header. The **source address** is the IP address of the send and the **destination address** is the IP address of the receiver.



Version

Version of IP Protocol. 4 and 6 are valid. This diagram represents version 4 structure only.

Header Length

Number of 32-bit words in TCP header, minimum value of 5. Multiply by 4 to get byte count.

Protocol

IP Protocol ID. Including (but not limited to):

1 ICMP	17 UDP	57 SKIP
2 IGMP	47 GRE	88 EIGRP
6 TCP	50 ESP	89 OSPF
9 IGRP	51 AH	115 L2TP

Total Length

Total length of IP datagram, or IP fragment if fragmented. Measured in Bytes.

Fragment Offset

Fragment offset from start of IP datagram. Measured in 8 byte (2 words, 64 bits) increments. If IP datagram is fragmented, fragment size (Total Length) must be a multiple of 8 bytes.

Header Checksum

Checksum of entire IP header

IP Flags

x D M

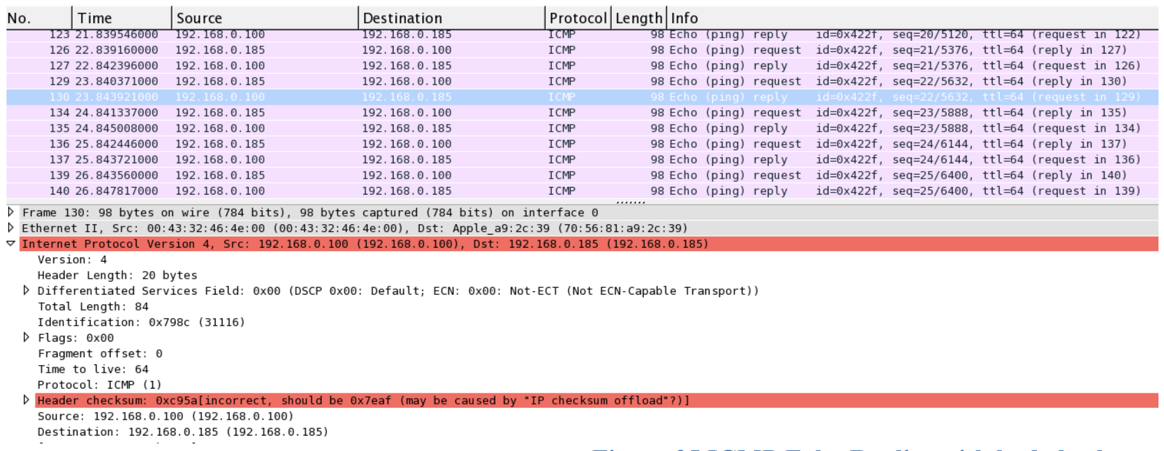
x 0x80 reserved (evil bit)
D 0x40 Do Not Fragment
M 0x20 More Fragments follow

RFC 791

Please refer to RFC 791 for the complete Internet Protocol (IP) Specification.

Figure 34 IP Header

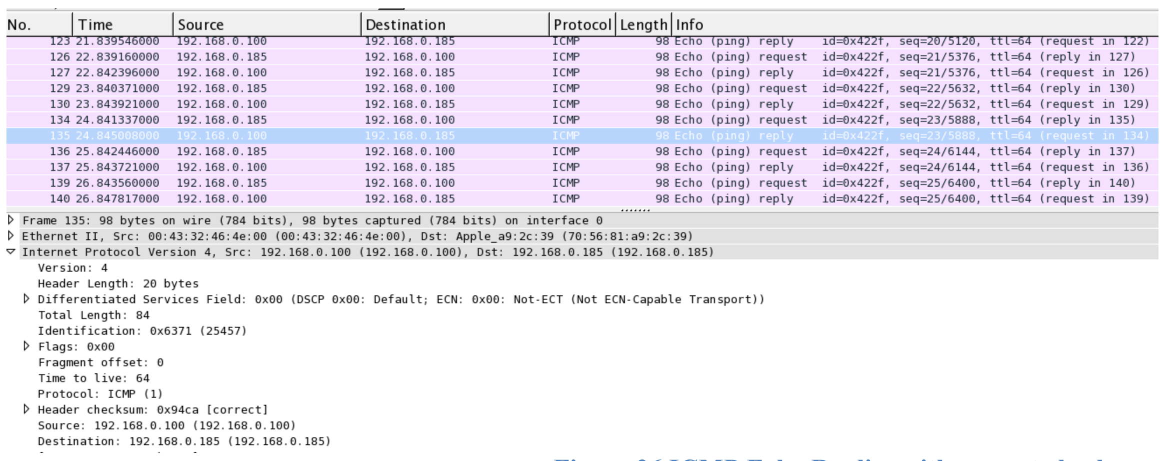
The new ping program, or the ICMP Echo reply, experienced issues with checksum. The error had to do with the one's complement function and the process of creating the function. The checksum had to be zeroed out first and the one's complement function needed to complement the result.



```
No.      Time              Source              Destination          Protocol  Length  Info
    123 21.839546000   192.168.0.100      192.168.0.185      ICMP        98 Echo (ping) reply    id=0x422f, seq=20/5120, ttl=64 (request in 122)
    126 22.839160000   192.168.0.185      192.168.0.100      ICMP        98 Echo (ping) request  id=0x422f, seq=21/5376, ttl=64 (reply in 127)
    127 22.842396000   192.168.0.100      192.168.0.185      ICMP        98 Echo (ping) reply    id=0x422f, seq=21/5376, ttl=64 (request in 126)
    129 23.840371000   192.168.0.185      192.168.0.100      ICMP        98 Echo (ping) request  id=0x422f, seq=22/5632, ttl=64 (reply in 130)
    130 23.843921000   192.168.0.100      192.168.0.185      ICMP        98 Echo (ping) reply    id=0x422f, seq=22/5632, ttl=64 (request in 129)
    134 24.841337000   192.168.0.185      192.168.0.100      ICMP        98 Echo (ping) request  id=0x422f, seq=23/5888, ttl=64 (reply in 135)
    135 24.845008000   192.168.0.100      192.168.0.185      ICMP        98 Echo (ping) reply    id=0x422f, seq=23/5888, ttl=64 (request in 134)
    136 25.842446000   192.168.0.185      192.168.0.100      ICMP        98 Echo (ping) request  id=0x422f, seq=24/6144, ttl=64 (reply in 137)
    137 25.843721000   192.168.0.100      192.168.0.185      ICMP        98 Echo (ping) reply    id=0x422f, seq=24/6144, ttl=64 (request in 136)
    139 26.843560000   192.168.0.185      192.168.0.100      ICMP        98 Echo (ping) request  id=0x422f, seq=25/6400, ttl=64 (reply in 140)
    140 26.847817000   192.168.0.100      192.168.0.185      ICMP        98 Echo (ping) reply    id=0x422f, seq=25/6400, ttl=64 (request in 139)

▷ Frame 130: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▷ Ethernet II, Src: 00:43:32:46:4e:00 (00:43:32:46:4e:00), Dst: Apple_a9:2c:39 (70:56:81:a9:2c:39)
▽ Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.185 (192.168.0.185)
    Version: 4
    Header Length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 84
    Identification: 0x798c (31116)
  ▷ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
  ▷ Header checksum: 0xc95a[incorrect, should be 0x7eaf (may be caused by "IP checksum offload"?)]
    Source: 192.168.0.100 (192.168.0.100)
    Destination: 192.168.0.185 (192.168.0.185)
```

Figure 35 ICMP Echo Replies with bad checksum



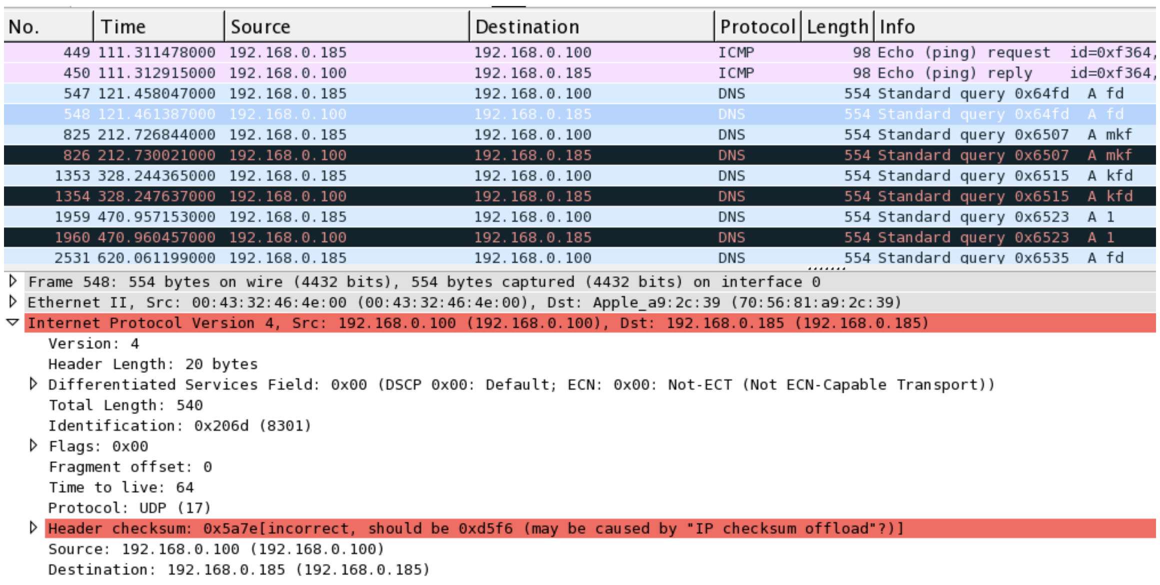
```
No.      Time              Source              Destination          Protocol  Length  Info
    123 21.839546000   192.168.0.100      192.168.0.185      ICMP        98 Echo (ping) reply    id=0x422f, seq=20/5120, ttl=64 (request in 122)
    126 22.839160000   192.168.0.185      192.168.0.100      ICMP        98 Echo (ping) request  id=0x422f, seq=21/5376, ttl=64 (reply in 127)
    127 22.842396000   192.168.0.100      192.168.0.185      ICMP        98 Echo (ping) reply    id=0x422f, seq=21/5376, ttl=64 (request in 126)
    129 23.840371000   192.168.0.185      192.168.0.100      ICMP        98 Echo (ping) request  id=0x422f, seq=22/5632, ttl=64 (reply in 130)
    130 23.843921000   192.168.0.100      192.168.0.185      ICMP        98 Echo (ping) reply    id=0x422f, seq=22/5632, ttl=64 (request in 129)
    134 24.841337000   192.168.0.185      192.168.0.100      ICMP        98 Echo (ping) request  id=0x422f, seq=23/5888, ttl=64 (reply in 135)
    135 24.845008000   192.168.0.100      192.168.0.185      ICMP        98 Echo (ping) reply    id=0x422f, seq=23/5888, ttl=64 (request in 134)
    136 25.842446000   192.168.0.185      192.168.0.100      ICMP        98 Echo (ping) request  id=0x422f, seq=24/6144, ttl=64 (reply in 137)
    137 25.843721000   192.168.0.100      192.168.0.185      ICMP        98 Echo (ping) reply    id=0x422f, seq=24/6144, ttl=64 (request in 136)
    139 26.843560000   192.168.0.185      192.168.0.100      ICMP        98 Echo (ping) request  id=0x422f, seq=25/6400, ttl=64 (reply in 140)
    140 26.847817000   192.168.0.100      192.168.0.185      ICMP        98 Echo (ping) reply    id=0x422f, seq=25/6400, ttl=64 (request in 139)

▷ Frame 135: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▷ Ethernet II, Src: 00:43:32:46:4e:00 (00:43:32:46:4e:00), Dst: Apple_a9:2c:39 (70:56:81:a9:2c:39)
▽ Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.185 (192.168.0.185)
    Version: 4
    Header Length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 84
    Identification: 0x6371 (25457)
  ▷ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
  ▷ Header checksum: 0x94ca [correct]
    Source: 192.168.0.100 (192.168.0.100)
    Destination: 192.168.0.185 (192.168.0.185)
```

Figure 36 ICMP Echo Replies with correct checksum

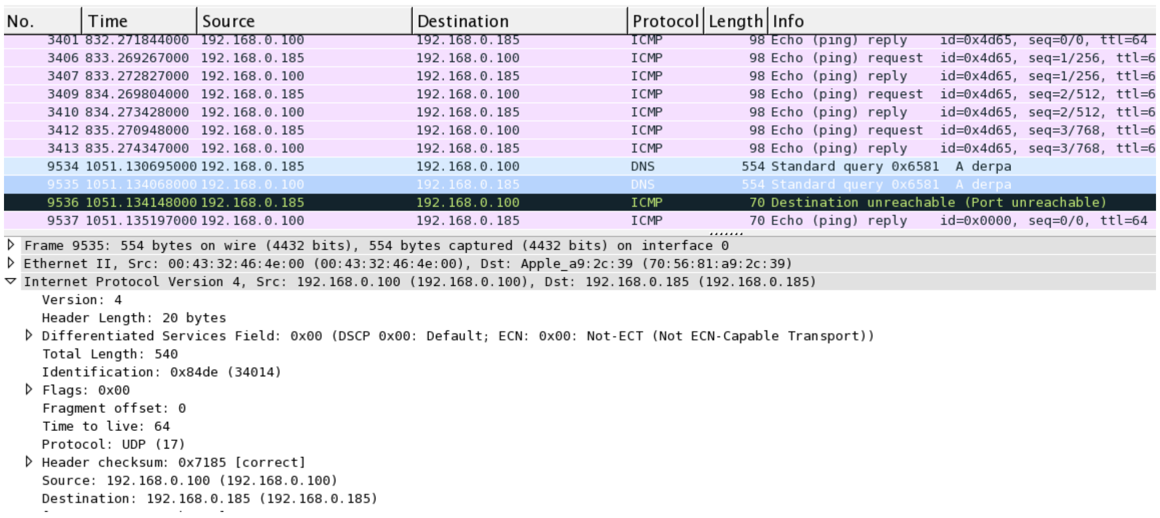
In order to debug the DNS server's failed responses; code was added in the IP parsing section in order to send the packet back to the host as is. That resulted in the discovery that the functions were not receiving the variable information properly. Reworking the code and how the passing of variables function to function was necessary to fix the DNS server code

Once the protocol header sections of the DNS server code had been debugged, the testing of the code corresponding to the DNS responses began. The DNS response code experienced the same issues with the IP checksum. The same solution with the ping program was applied to the DNS server code, which fixed the checksum issue here as well.



```
No.   Time           Source         Destination    Protocol Length Info
  449 111.311478000  192.168.0.185  192.168.0.100  ICMP         98 Echo (ping) request  id=0xf364,
  450 111.312915000  192.168.0.100  192.168.0.185  ICMP         98 Echo (ping) reply    id=0xf364,
  547 121.458047000  192.168.0.185  192.168.0.100  DNS         554 Standard query 0x64fd  A fd
  548 121.461387000  192.168.0.100  192.168.0.185  DNS         554 Standard query 0x64fd  A fd
  825 212.726844000  192.168.0.185  192.168.0.100  DNS         554 Standard query 0x6507  A mkf
  826 212.730021000  192.168.0.100  192.168.0.185  DNS         554 Standard query 0x6507  A mkf
 1353 328.244365000  192.168.0.185  192.168.0.100  DNS         554 Standard query 0x6515  A kfd
 1354 328.247637000  192.168.0.100  192.168.0.185  DNS         554 Standard query 0x6515  A kfd
 1959 470.957153000  192.168.0.185  192.168.0.100  DNS         554 Standard query 0x6523  A 1
 1960 470.960457000  192.168.0.100  192.168.0.185  DNS         554 Standard query 0x6523  A 1
 2531 620.061199000  192.168.0.185  192.168.0.100  DNS         554 Standard query 0x6535  A fd
▷ Frame 548: 554 bytes on wire (4432 bits), 554 bytes captured (4432 bits) on interface 0
▷ Ethernet II, Src: 00:43:32:46:4e:00 (00:43:32:46:4e:00), Dst: Apple_a9:2c:39 (70:56:81:a9:2c:39)
▽ Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.185 (192.168.0.185)
    Version: 4
    Header Length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 540
    Identification: 0x206d (8301)
  ▷ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (17)
  ▷ Header checksum: 0x5a7e[incorrect, should be 0xd5f6 (may be caused by "IP checksum offload"?)]
    Source: 192.168.0.100 (192.168.0.100)
    Destination: 192.168.0.185 (192.168.0.185)
```

Figure 37 NetFPGA DNS return query bad checksum



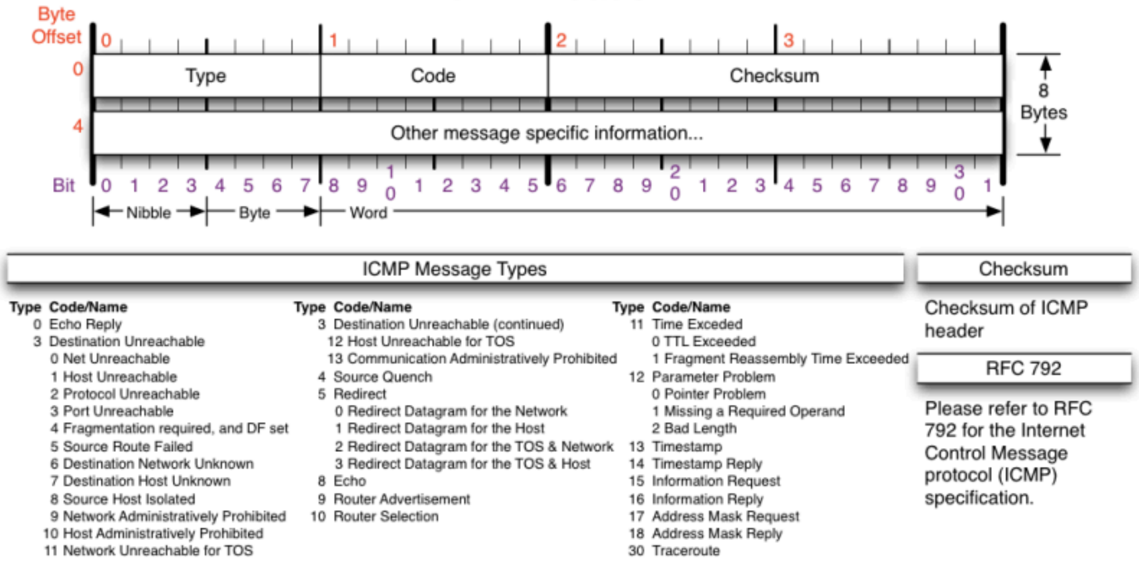
```
No.   Time            Source         Destination    Protocol Length Info
 3401 832.271844000   192.168.0.100  192.168.0.185  ICMP         98 Echo (ping) reply    id=0x4d65, seq=0/0, ttl=64
 3406 833.269267000   192.168.0.185  192.168.0.100  ICMP         98 Echo (ping) request  id=0x4d65, seq=1/256, ttl=6
 3407 833.272827000   192.168.0.100  192.168.0.185  ICMP         98 Echo (ping) reply    id=0x4d65, seq=1/256, ttl=6
 3409 834.269804000   192.168.0.185  192.168.0.100  ICMP         98 Echo (ping) request  id=0x4d65, seq=2/512, ttl=6
 3410 834.273428000   192.168.0.100  192.168.0.185  ICMP         98 Echo (ping) reply    id=0x4d65, seq=2/512, ttl=6
 3412 835.270948000   192.168.0.185  192.168.0.100  ICMP         98 Echo (ping) request  id=0x4d65, seq=3/768, ttl=6
 3413 835.274347000   192.168.0.100  192.168.0.185  ICMP         98 Echo (ping) reply    id=0x4d65, seq=3/768, ttl=6
 9534 1051.130695000  192.168.0.185  192.168.0.100  DNS         554 Standard query 0x6581  A derpa
 9535 1051.134068000  192.168.0.100  192.168.0.185  DNS         554 Standard query 0x6581  A derpa
 9536 1051.134148000  192.168.0.185  192.168.0.100  ICMP         70 Destination unreachable (Port unreachable)
 9537 1051.135197000  192.168.0.100  192.168.0.185  ICMP         70 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64
▷ Frame 9535: 554 bytes on wire (4432 bits), 554 bytes captured (4432 bits) on interface 0
▷ Ethernet II, Src: 00:43:32:46:4e:00 (00:43:32:46:4e:00), Dst: Apple_a9:2c:39 (70:56:81:a9:2c:39)
▽ Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.185 (192.168.0.185)
    Version: 4
    Header Length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 540
    Identification: 0x84de (34014)
  ▷ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: UDP (17)
  ▷ Header checksum: 0x7185 [correct]
    Source: 192.168.0.100 (192.168.0.100)
    Destination: 192.168.0.185 (192.168.0.185)
```

Figure 38 NetFPGA DNS return query good checksum

4.3.6 ICMP: Internet Control Message Protocol: network layer (layer 3)

Internet Control Message Protocol (ICMP) is a network layer protocol that is used for debugging of various networking situations. ICMP is also used to help report errors in datagram processing. There are various types of ICMP messages; however, in the NetFPGA DNS design, the Echo reply was the only type implemented. The use of the ping program was to perform an ICMP Echo request to the NetFPGA to ensure connectivity was established. This greatly helped in discovering the issues of the network not knowing about the NetFPGA, as it was a simple test for connectivity. Furthermore the use of the ICMP Echo request/reply aided in determining that connectivity to the NetFPGA was maintained (as occasionally the code would enter a state in which the NetFPGA was hung and required the code reloaded).

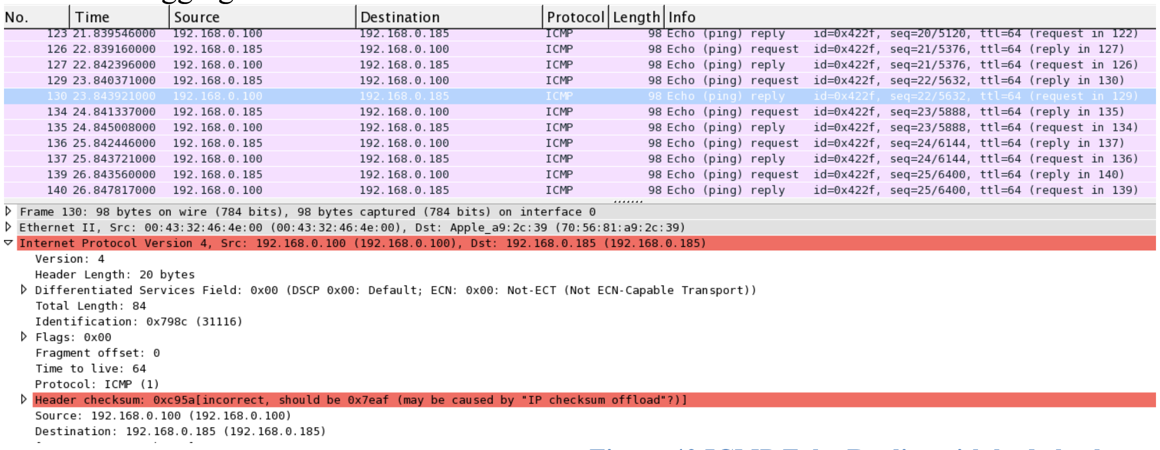
The **type** is an unsigned 8-bit integer and the values that are used in the DNS server code are 8 to check if the packet is an Echo request and 0 to signify the packet sent from the NetFPGA is an Echo reply. The **code** is an unsigned 8-bit integer for error reporting, though with the server only using Echo replies, 0 is the code used in the NetFPGA DNS server. The **checksum** is an unsigned 16-bit integer and is a one's complement of the ICMP message. The **other message specification** is divided into an unsigned 16-bit integer **identifier** and an unsigned 16-bit integer **sequence number**. The **identifier** and **sequence number** are used to match requests and replies. In the DNS server the only field changed in ICMP messages is the type, the rest are copied from the Echo request.


Byte Offset
0
4
Type
Code
Checksum
Other message specific information...
8 Bytes
Bit
Nibble
Byte
Word
ICMP Message Types
Checksum
Checksum of ICMP header
RFC 792
Please refer to RFC 792 for the Internet Control Message protocol (ICMP) specification.

Type	Code/Name
0	Echo Reply
3	Destination Unreachable
	0 Net Unreachable
	1 Host Unreachable
	2 Protocol Unreachable
	3 Port Unreachable
	4 Fragmentation required, and DF set
	5 Source Route Failed
	6 Destination Network Unknown
	7 Destination Host Unknown
	8 Source Host Isolated
	9 Network Administratively Prohibited
	10 Host Administratively Prohibited
	11 Network Unreachable for TOS
3	Destination Unreachable (continued)
	12 Host Unreachable for TOS
	13 Communication Administratively Prohibited
4	Source Quench
5	Redirect
	0 Redirect Datagram for the Network
	1 Redirect Datagram for the Host
	2 Redirect Datagram for the TOS & Network
	3 Redirect Datagram for the TOS & Host
8	Echo
9	Router Advertisement
10	Router Selection
11	Time Exceded
	0 TTL Exceeded
	1 Fragment Reassembly Time Exceeded
12	Parameter Problem
	0 Pointer Problem
	1 Missing a Required Operand
	2 Bad Length
13	Timestamp
14	Timestamp Reply
15	Information Request
16	Information Reply
17	Address Mask Request
18	Address Mask Reply
30	Traceroute

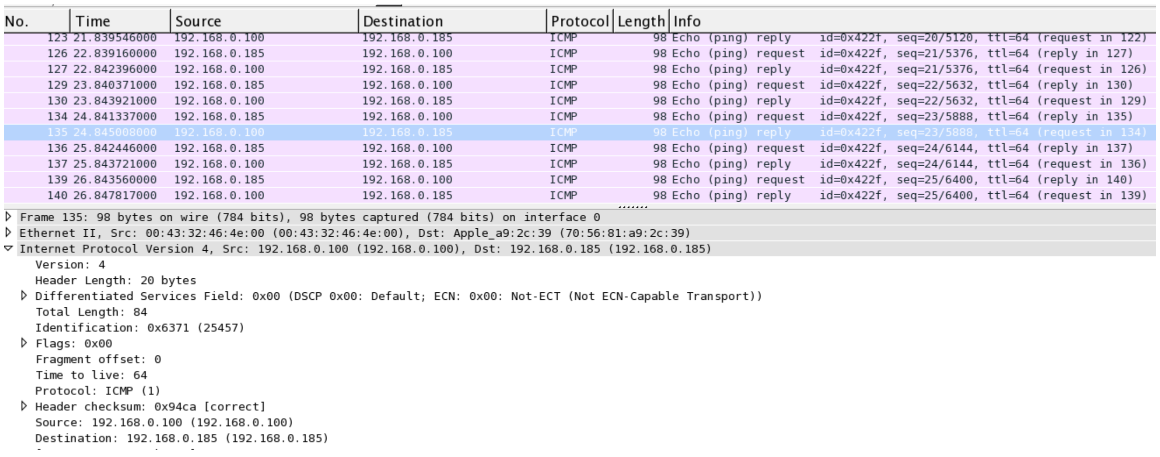
Figure 39 ICMP Header

Once the ARP issue was discovered and solution found (see section on ARP) a new ping program was created. The new ping program began replying to the ICMP Echo requests; however, the checksum was not calculated correctly. The checksum function was reworked, and the process to calculate the checksum was established and the new ping program was successful. The ICMP Echo replies were added to aid in debugging network connectivity. Though, the DNS server was still unresponsive and required further debugging.



```
No.   Time          Source          Destination     Protocol Length Info
  123 21.839546000  192.168.0.100   192.168.0.185   ICMP         98 Echo (ping) reply    id=0x422f, seq=20/5120, ttl=64 (request in 122)
  126 22.839160000  192.168.0.185   192.168.0.100   ICMP         98 Echo (ping) request  id=0x422f, seq=21/5376, ttl=64 (reply in 127)
  127 22.842396000  192.168.0.100   192.168.0.185   ICMP         98 Echo (ping) reply    id=0x422f, seq=21/5376, ttl=64 (request in 126)
  129 23.840371000  192.168.0.185   192.168.0.100   ICMP         98 Echo (ping) request  id=0x422f, seq=22/5632, ttl=64 (reply in 130)
  130 23.843921000  192.168.0.100   192.168.0.185   ICMP         98 Echo (ping) reply    id=0x422f, seq=22/5632, ttl=64 (request in 129)
  134 24.841337000  192.168.0.185   192.168.0.100   ICMP         98 Echo (ping) request  id=0x422f, seq=23/5888, ttl=64 (reply in 135)
  135 24.845008000  192.168.0.100   192.168.0.185   ICMP         98 Echo (ping) reply    id=0x422f, seq=23/5888, ttl=64 (request in 134)
  136 25.842446000  192.168.0.185   192.168.0.100   ICMP         98 Echo (ping) request  id=0x422f, seq=24/6144, ttl=64 (reply in 137)
  137 25.843721000  192.168.0.100   192.168.0.185   ICMP         98 Echo (ping) reply    id=0x422f, seq=24/6144, ttl=64 (request in 136)
  139 26.843560000  192.168.0.185   192.168.0.100   ICMP         98 Echo (ping) request  id=0x422f, seq=25/6400, ttl=64 (reply in 140)
  140 26.847817000  192.168.0.100   192.168.0.185   ICMP         98 Echo (ping) reply    id=0x422f, seq=25/6400, ttl=64 (request in 139)
▷ Frame 130: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▷ Ethernet II, Src: 00:43:32:46:4e:00 (00:43:32:46:4e:00), Dst: Apple_a9:2c:39 (70:56:81:a9:2c:39)
▽ Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.185 (192.168.0.185)
    Version: 4
    Header Length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 84
    Identification: 0x798c (31116)
  ▷ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
  ▷ Header checksum: 0xc95a[incorrect, should be 0x7eaf (may be caused by "IP checksum offload"?)]
    Source: 192.168.0.100 (192.168.0.100)
    Destination: 192.168.0.185 (192.168.0.185)
```

Figure 40 ICMP Echo Replies with bad checksum



```
No.   Time         Source          Destination     Protocol Length Info
  123 21.839546000 192.168.0.100   192.168.0.185   ICMP        98 Echo (ping) reply    id=0x422f, seq=20/5120, ttl=64 (request in 122)
  126 22.839160000 192.168.0.185   192.168.0.100   ICMP        98 Echo (ping) request  id=0x422f, seq=21/5376, ttl=64 (reply in 127)
  127 22.842396000 192.168.0.100   192.168.0.185   ICMP        98 Echo (ping) reply    id=0x422f, seq=21/5376, ttl=64 (request in 126)
  129 23.840371000 192.168.0.185   192.168.0.100   ICMP        98 Echo (ping) request  id=0x422f, seq=22/5632, ttl=64 (reply in 130)
  130 23.843921000 192.168.0.100   192.168.0.185   ICMP        98 Echo (ping) reply    id=0x422f, seq=22/5632, ttl=64 (request in 129)
  134 24.841337000 192.168.0.185   192.168.0.100   ICMP        98 Echo (ping) request  id=0x422f, seq=23/5888, ttl=64 (reply in 135)
  135 24.845008000 192.168.0.100   192.168.0.185   ICMP        98 Echo (ping) reply    id=0x422f, seq=23/5888, ttl=64 (request in 134)
  136 25.842446000 192.168.0.185   192.168.0.100   ICMP        98 Echo (ping) request  id=0x422f, seq=24/6144, ttl=64 (reply in 137)
  137 25.843721000 192.168.0.100   192.168.0.185   ICMP        98 Echo (ping) reply    id=0x422f, seq=24/6144, ttl=64 (request in 136)
  139 26.843560000 192.168.0.185   192.168.0.100   ICMP        98 Echo (ping) request  id=0x422f, seq=25/6400, ttl=64 (reply in 140)
  140 26.847817000 192.168.0.100   192.168.0.185   ICMP        98 Echo (ping) reply    id=0x422f, seq=25/6400, ttl=64 (request in 139)

▷ Frame 135: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▷ Ethernet II, Src: 00:43:32:46:4e:00 (00:43:32:46:4e:00), Dst: Apple_a9:2c:39 (70:56:81:a9:2c:39)
▽ Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.185 (192.168.0.185)
    Version: 4
    Header Length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 84
    Identification: 0x6371 (25457)
  ▷ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
  ▷ Header checksum: 0x94ca [correct]
    Source: 192.168.0.100 (192.168.0.100)
    Destination: 192.168.0.185 (192.168.0.185)
```

Figure 41 ICMP Echo Replies with correct checksum

4.3.7 UDP: User Datagram Protocol: transport layer (layer 4)

User Datagram Protocol (UDP) is a transport layer protocol. It is a connectionless protocol that facilitates transfer of data across the network. The protocol does not provide reliability from packet loss, unlike TCP, and relies on the overarching program to provide reliability from packet loss. Often the choice to use UDP occurs in situations where packet loss is acceptable. UDP is the typical transport protocol that encapsulates DNS packets.

DNS requests utilize UDP as the transport protocol as packets loss preferred as opposed to overburdening the server with the TCP retransmission architecture. The exception to that is a zone transfer between DNS Servers, due to the importance of the integrity of the zone files. However, since the project design was not intended for zone transfers to be done by the NetFPGA, only UDP was implemented on the NetFPGA.

The UDP **source port** is the port of the sender and the **destination port** is the port of the receiver. The DNS server listens on port 53, so if the resolver is sending the packet, the **destination port** will be 53 and the **source port** can be any port the host chooses, typically a port that does not have an application already associated with it. The port range 49152-65535 is available for dynamic use, so a host would chose one of those ports. The DNS server would send the packet back to the resolver using the port it received the packet as the **destination port** and the **source port** being 53. The **length** is the length of the UDP data with the size of the UDP header. The **checksum** is a one's complement calculation, like the IP checksum; however, it uses the UDP packet (header and data) and a pseudo-header (IP source address, IP destination address, IP protocol, and the UDP length).

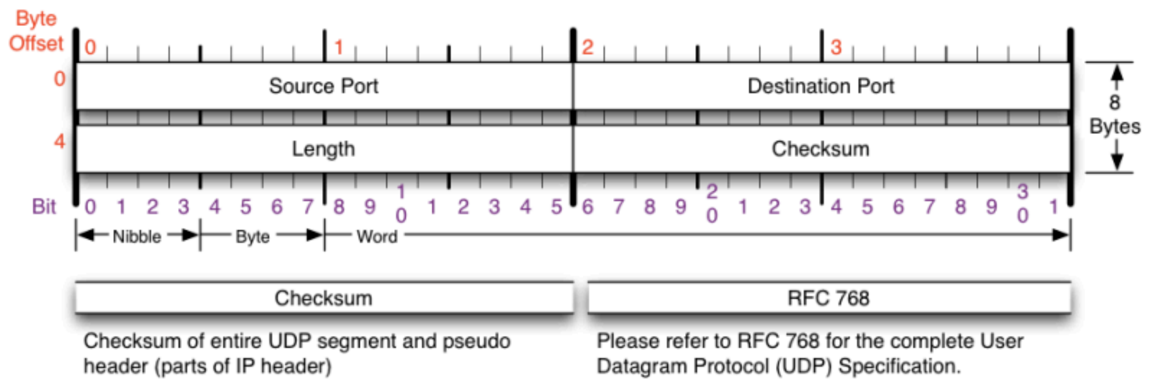
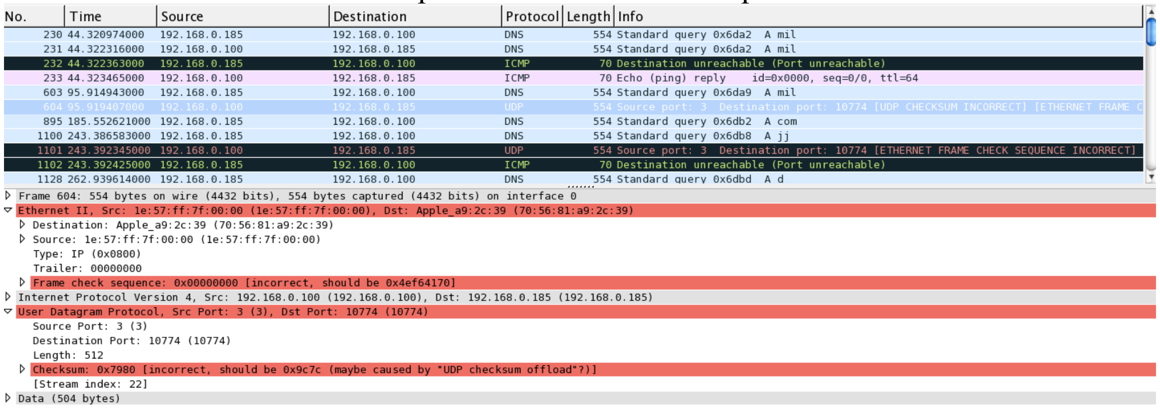


Figure 42 UDP Header

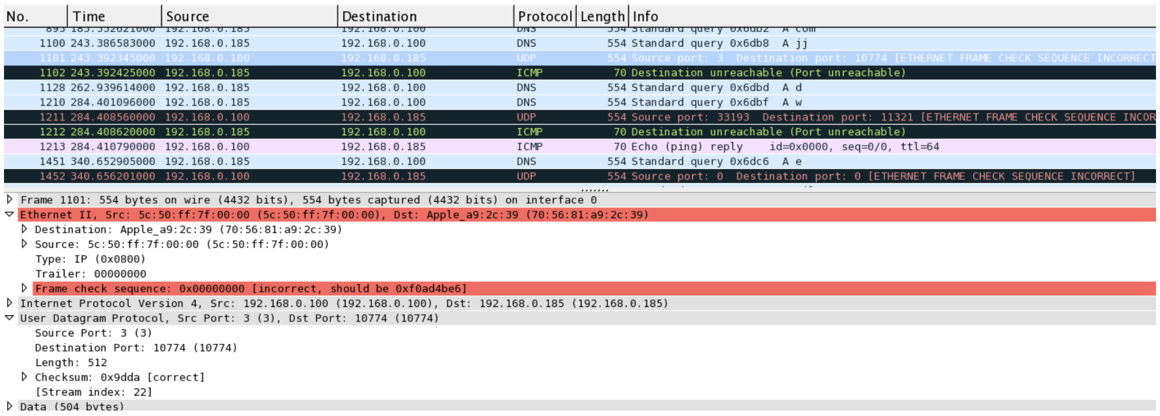
The UDP checksum is calculated differently than the IP checksum. The process had to be reworked and some separate functions created in order to properly calculate the UDP checksum. The functions created followed the same process as the IP checksum; however, it required that it be broken into separate functions. The functions were to calculate the one's complement and the other was to fold it into a 16-bit unsigned integer. The UDP checksum calculation process was reworked to provide the correct value.



```
No.   Time            Source          Destination     Protocol Length Info
  230 44.320974000    192.168.0.185   192.168.0.100   DNS      554 Standard query 0x6da2  A mil
  231 44.322316000    192.168.0.100   192.168.0.185   DNS      554 Standard query 0x6da2  A mil
  232 44.322363000    192.168.0.185   192.168.0.100   ICMP      70 Destination unreachable (Port unreachable)
  233 44.323465000    192.168.0.100   192.168.0.185   ICMP      70 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64
  603 95.914943000    192.168.0.185   192.168.0.100   DNS      554 Standard query 0x6da9  A mil
  604 95.919407000    192.168.0.100   192.168.0.185   UDP      554 Source port: 3  Destination port: 10774 [UDP CHECKSUM INCORRECT] [ETHERNET FRAME C
  895 185.552621000   192.168.0.185   192.168.0.100   DNS      554 Standard query 0x6db2  A com
 1100 243.386583000   192.168.0.185   192.168.0.100   DNS      554 Standard query 0x6db8  A jj
 1101 243.392345000   192.168.0.100   192.168.0.185   UDP      554 Source port: 3  Destination port: 10774 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
 1102 243.392425000   192.168.0.185   192.168.0.100   ICMP      70 Destination unreachable (Port unreachable)
 1128 262.939614000   192.168.0.185   192.168.0.100   DNS      554 Standard query 0x6dbd  A d

▷ Frame 604: 554 bytes on wire (4432 bits), 554 bytes captured (4432 bits) on interface 0
▽ Ethernet II, Src: 1e:57:ff:7f:00:00 (1e:57:ff:7f:00:00), Dst: Apple_a9:2c:39 (70:56:81:a9:2c:39)
  ▷ Destination: Apple_a9:2c:39 (70:56:81:a9:2c:39)
  ▷ Source: 1e:57:ff:7f:00:00 (1e:57:ff:7f:00:00)
    Type: IP (0x0800)
    Trailer: 00000000
  ▷ Frame check sequence: 0x00000000 [incorrect, should be 0x4ef64170]
▷ Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.185 (192.168.0.185)
▽ User Datagram Protocol, Src Port: 3 (3), Dst Port: 10774 (10774)
    Source Port: 3 (3)
    Destination Port: 10774 (10774)
    Length: 512
  ▷ Checksum: 0x7980 [incorrect, should be 0x9c7c (maybe caused by "UDP checksum offload"?)]
    [Stream index: 22]
▷ Data (504 bytes)
```

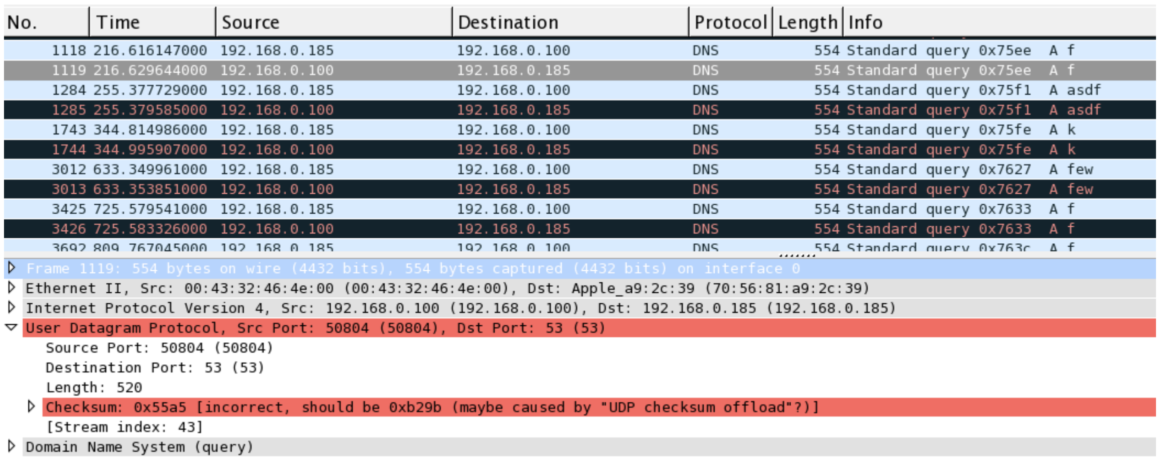
Figure 43 NetFPGA DNS response bad Ethernet header bad UDP checksum



```
No.   Time            Source          Destination     Protocol Length Info
  895 185.552621000   192.168.0.185   192.168.0.100   DNS      554 Standard query 0x6db2  A com
 1100 243.386583000   192.168.0.185   192.168.0.100   DNS      554 Standard query 0x6db8  A jj
 1101 243.392345000   192.168.0.100   192.168.0.185   UDP      554 Source port: 3  Destination port: 10774 [ETHERNET FRAME CHECK SEQUENCE INCORRECT
 1102 243.392425000   192.168.0.185   192.168.0.100   ICMP      70 Destination unreachable (Port unreachable)
 1128 262.939614000   192.168.0.185   192.168.0.100   DNS      554 Standard query 0x6dbd  A d
 1210 284.401096000   192.168.0.185   192.168.0.100   DNS      554 Standard query 0x6dbf  A w
 1211 284.408560000   192.168.0.100   192.168.0.185   UDP      554 Source port: 33193  Destination port: 11321 [ETHERNET FRAME CHECK SEQUENCE INCOR
 1212 284.408620000   192.168.0.185   192.168.0.100   ICMP      70 Destination unreachable (Port unreachable)
 1213 284.410790000   192.168.0.100   192.168.0.185   ICMP      70 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64
 1451 340.652905000   192.168.0.185   192.168.0.100   DNS      554 Standard query 0x6dc6  A e
 1452 340.656201000   192.168.0.100   192.168.0.185   UDP      554 Source port: 0  Destination port: 0 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]

▷ Frame 1101: 554 bytes on wire (4432 bits), 554 bytes captured (4432 bits) on interface 0
▽ Ethernet II, Src: 5c:50:ff:7f:00:00 (5c:50:ff:7f:00:00), Dst: Apple_a9:2c:39 (70:56:81:a9:2c:39)
  ▷ Destination: Apple_a9:2c:39 (70:56:81:a9:2c:39)
  ▷ Source: 5c:50:ff:7f:00:00 (5c:50:ff:7f:00:00)
    Type: IP (0x0800)
    Trailer: 00000000
  ▷ Frame check sequence: 0x00000000 [incorrect, should be 0xf0ad4be6]
▷ Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.185 (192.168.0.185)
▽ User Datagram Protocol, Src Port: 3 (3), Dst Port: 10774 (10774)
    Source Port: 3 (3)
    Destination Port: 10774 (10774)
    Length: 512
  ▷ Checksum: 0x9dda [correct]
    [Stream index: 22]
▷ Data (504 bytes)
```

Figure 44 NetFPGA DNS response bad Ethernet header good UDP checksum

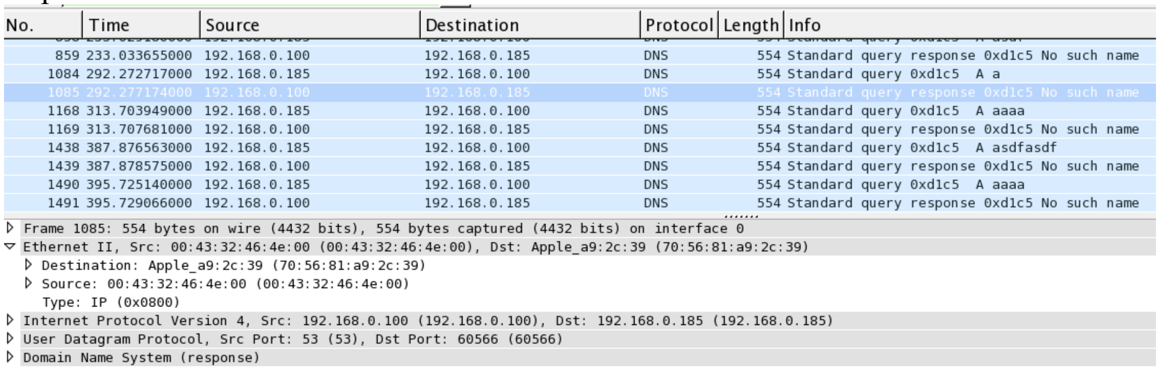


```
No.   Time            Source          Destination     Protocol Length Info
 1118 216.616147000  192.168.0.185   192.168.0.100   DNS        554 Standard query 0x75ee  A f
 1119 216.629644000  192.168.0.100   192.168.0.185   DNS        554 Standard query 0x75ee  A f
 1284 255.377729000  192.168.0.185   192.168.0.100   DNS        554 Standard query 0x75f1  A asdf
 1285 255.379585000  192.168.0.100   192.168.0.185   DNS        554 Standard query 0x75f1  A asdf
 1743 344.814986000  192.168.0.185   192.168.0.100   DNS        554 Standard query 0x75fe  A k
 1744 344.995907000  192.168.0.100   192.168.0.185   DNS        554 Standard query 0x75fe  A k
 3012 633.349961000  192.168.0.185   192.168.0.100   DNS        554 Standard query 0x7627  A few
 3013 633.353851000  192.168.0.100   192.168.0.185   DNS        554 Standard query 0x7627  A few
 3425 725.579541000  192.168.0.185   192.168.0.100   DNS        554 Standard query 0x7633  A f
 3426 725.583326000  192.168.0.100   192.168.0.185   DNS        554 Standard query 0x7633  A f
 3692 809.767045000  192.168.0.185   192.168.0.100   DNS        554 Standard query 0x763c  A f
▷ Frame 1119: 554 bytes on wire (4432 bits), 554 bytes captured (4432 bits) on interface 0
▷ Ethernet II, Src: 00:43:32:46:4e:00 (00:43:32:46:4e:00), Dst: Apple_a9:2c:39 (70:56:81:a9:2c:39)
▷ Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.185 (192.168.0.185)
▽ User Datagram Protocol, Src Port: 50804 (50804), Dst Port: 53 (53)
    Source Port: 50804 (50804)
    Destination Port: 53 (53)
    Length: 520
  ▷ Checksum: 0x55a5 [incorrect, should be 0xb29b (maybe caused by "UDP checksum offload"?)]
    [Stream index: 43]
▷ Domain Name System (query)
```

Figure 45 NetFPGA DNS response bad UDP checksum

4.3.8 DNS Protocol: Domain Name System Protocol: application layer (layer 7)

The DNS protocol is an application layer protocol and has been explained extensively in previous sections. The NetFPGA DNS server successfully responded to the DNS query using the NetThreads compiler project. Some examples of the successful responses are shown below.



```
No.   Time            Source          Destination     Protocol Length Info
  859 233.033655000  192.168.0.100   192.168.0.185   DNS        554 Standard query response 0xd1c5 No such name
 1084 292.272717000  192.168.0.185   192.168.0.100   DNS        554 Standard query 0xd1c5  A a
 1085 292.277174000  192.168.0.100   192.168.0.185   DNS        554 Standard query response 0xd1c5 No such name
 1168 313.703949000  192.168.0.185   192.168.0.100   DNS        554 Standard query 0xd1c5  A aaaa
 1169 313.707681000  192.168.0.100   192.168.0.185   DNS        554 Standard query response 0xd1c5 No such name
 1438 387.876563000  192.168.0.185   192.168.0.100   DNS        554 Standard query 0xd1c5  A asdfasdf
 1439 387.878575000  192.168.0.100   192.168.0.185   DNS        554 Standard query response 0xd1c5 No such name
 1490 395.725140000  192.168.0.185   192.168.0.100   DNS        554 Standard query 0xd1c5  A aaaa
 1491 395.729066000  192.168.0.100   192.168.0.185   DNS        554 Standard query response 0xd1c5 No such name
▷ Frame 1085: 554 bytes on wire (4432 bits), 554 bytes captured (4432 bits) on interface 0
▽ Ethernet II, Src: 00:43:32:46:4e:00 (00:43:32:46:4e:00), Dst: Apple_a9:2c:39 (70:56:81:a9:2c:39)
  ▷ Destination: Apple_a9:2c:39 (70:56:81:a9:2c:39)
  ▷ Source: 00:43:32:46:4e:00 (00:43:32:46:4e:00)
    Type: IP (0x0800)
▷ Internet Protocol Version 4, Src: 192.168.0.100 (192.168.0.100), Dst: 192.168.0.185 (192.168.0.185)
▷ User Datagram Protocol, Src Port: 53 (53), Dst Port: 60566 (60566)
▷ Domain Name System (response)
```

Figure 46 NetFPGA DNS response shown with Ethernet section

While the responses were successful, those responses were not very useful since the lookups were unable to be performed. There were a couple attempts made to load the lookup structure and both were unsuccessful. One attempt was to load the Trie lookup structure from an external C code into the NetFPGA's DRAM memory. The other attempt was to have the lookup structure loaded into the NetFPGA's DRAM memory using a generated NetThreads specific function.

The idea to load the Trie lookup structure into the NetFPGA from external C code was inspired by the HighPerformancePacketClassifier NetFPGA project and utilizes function from that project. In order to load the items into NetFPGA memory using the functions for the HighPerformancePacketClassifier project, the memory address needs to be specified. This meant the memory addresses need to be calculated for the values of the Trie lookup structure after the Master files was parsed. The C code created to perform those tasks did indeed load the Trie lookup structure onto the NetFPGA. However, when the NetThreads bitfile was loaded, the code appeared to have failed to load the structure.

The second and last attempt was to create a C function to load the structure from the Master file from a generated header file. The generated function would load the structure using calls to *sp_malloc*, the NetThreads version of *malloc* that allocates memory space in the NetFPGA DRAM, and *memcpy* to put the values into memory. This code was tested with the software version by modifying the generator code to use *malloc* instead of *sp_malloc* and by modifying the software version to use the generated function. In the modified software version, the generated function worked in loading the Trie lookup structure. However, when using the NetThreads friendly generated function in the NetThreads DNS server it failed. There are multiple reasons why this might have failed which would require a better system of debugging. Also, a better understanding of the limitation of the NetThreads C could aid in the reasons why the loading of the Trie lookup structure failed.

5 Conclusion

The NetFPGA DNS server did successfully respond to queries and a hardware DNS server was created. Although, a hardware DNS server was created, it was not a useful DNS server, since the lookups were unable to be performed. Once some of the initial issues were worked through, the NetThreads compiler did make it easier to code for the NetFPGA. The debugging of issues is painful, since the debug process of the NetThreads project had issues. Also, there are still some unknowns in relation to how the NetThreads compiler works, such as the issues experienced with the Trie lookup structure. A better method of debugging would make the NetThreads compiler a great NetFPGA prototyping tool.

A Verilog solution would be a better approach to take this project forward. Though the NetThreads compiler took out many of the difficulties of working with Verilog, such as timing and working at that low level of programming, the unknowns of NetThreads and the lack of efficient debugging outweigh the benefits. The tools to simulate the design would aid significantly to help in debugging issues that would arise. Using Verilog results in designs that offer more control than that of the NetThreads compiler, making the potential for better optimizations possible.

6 References

Antichi, Gianni. "High Performance Packet Classifier." GitHub. 22 Mar. 2011. Web. <https://github.com/Caustic/netfpga-wiki/wiki/HighPerformancePacketClassifier>.

Baboescu, Florin, Dean M. Tullsen, Grigore Rosu, Sumeet Singh. “A Tree Based Router Search Engine Architecture With Single Port Memories (ISCA’05), 2005.

Baxter, Matt. "TCP/IP Reference." Nmap. Web. <http://nmap.org/book/tcpip-ref.html>. <http://nmap.org/book/images/hdr/MJB-IP-Header-800x576.png>. <http://nmap.org/book/images/hdr/MJB-ICMP-Header-800x392.png>. <http://nmap.org/book/images/hdr/MJB-UDP-Header-800x264.png>.

Becchi, Michela. “Introduction to NetFPGAs”. ECE4280/7280 Network Systems Architecture. June 2015. Slides.

"Developer's Guide." GitHub. 26 Feb. 2013. Web. <https://github.com/NetFPGA/netfpga/wiki/DevelopersGuide>.

"Ethernet (IEEE 802.3)." WireShark. Web. <https://wiki.wireshark.org/Ethernet>.

"Internet Protocol DARPA Internet Program Protocol Specification", RFC 791, University of Southern California /Information Sciences Institute, September 1981, <http://www.ietf.org/rfc/rfc791.txt>.

Jiang, Weirong, Viktor K Prasanna. "A MemoryBalanced Linear Pipeline Architecture for Triebased IP Lookup." in Proc. HotI '07, 2007.

J. Postel, J. Reynolds, "A Standard for the Transmission of IP Datagrams over IEEE 802 Networks", RFC 1042, University of Southern California /Information Sciences Institute, February 1988, < http://www.ietf.org/rfc/rfc1042.txt>.

J. Postel, "Internet Control Message Protocol DARPA Internet Program Protocol Specification", RFC 792, University of Southern California /Information Sciences Institute, September 1981, <http://www.ietf.org/rfc/rfc792.txt>.

J. Postel, "User Datagram Protocol", RFC 768, University of Southern California /Information Sciences Institute, August 28 1980, <http://www.ietf.org/rfc/rfc768.txt>.

Khalid, Ahmed. High Speed NetFPGA Router A Step by Step Guide on Developing a High Speed Router on NetFPGA Board. Saarbrücken: Lambert Academic, 2012. Print.

Kumar, Sailesh , Michela Becchi, Patrick Crowly, Jonathan Turner, "CAMP: Fast and Efficient IP Lookup Architecture", *ANCS '06,* pp.51–60, December 2006, <http://dl.acm.org/citation.cfm?id=1185347.1185355&coll=DL&dl=ACM&CFID=329255860&CFTOKEN=70874187>.

Kurose, James F., and Keith W. Ross. Computer Networking A Top-Down Approach Featuring the Internet. 3rd ed. Boston: Pearson Education, 2005. Print.

Labrecque, Martin, J. Gregory Steffan, Geoffrey Salmon, Monia Ghobadi, and Yashar Ganjali. "NetThreads: Programming NetFPGA with Threaded Software." (2009). Web.

Mockapetris, Paul, "Domain Names: Concepts and Facilities", RFC 1034, University of Southern California/Information Sciences Institute, November 1987, <http://www.ietf.org/rfc/rfc1034.txt>.

Mockapetris, Paul, "Domain Names: Implementation and Specification", RFC 1035, University of Southern California /Information Sciences Institute, November 1987, <http://www.ietf.org/rfc/rfc1035.txt>.

M. Lottor, "Domain Administrators Operations Guide", RFC 1033, SRI International, November 1987, < http://www.ietf.org/rfc/rfc1033.txt>.

"NetFPGA 1G." NetFPGA. Web. < http://netfpga.org/site/#/systems/4netfpga-1g/details/>.

"NetFPGA Guide." GitHub. 28 Apr. 2013. Web. <https://github.com/NetFPGA/netfpga/wiki/Guide>.

"NetThreads." GitHub. 27 Feb. 2013. Web. <https://github.com/NetFPGA/netfpga/wiki/NetThreads>.

Peterson, Larry L., and Bruce S. Davie. Computer Networks A Systems Approach. 4th ed. San Francisco: Morgan Kaufmann, 2007. Print.

Plummer, David C., "An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware", RFC 826, November 1982, < http://www.ietf.org/rfc/rfc826.txt>.

R. Braden, “Requirements for Internet Hosts – Communication Layers”, RFC 1122, Internet Engineering Task Force, October 1989, <http://www.ietf.org/rfc/rfc1122.txt>.

"Reference NIC Walkthrough." GitHub. 14 Jan. 2013. Web. <https://github.com/NetFPGA/netfpga/wiki/ReferenceNICWalkthrough#Reference_Pipeline>.

"Reference Router Walkthrough." GitHub. 14 Jan. 2013. Web. <https://github.com/NetFPGA/netfpga/wiki/ReferenceRouterWalkthrough>.

Reforgiato, Diego, and Fabio Battaglia. NetFGPA Architecture and Hardware Description An Insight of the NetFPGA Platform. Saarbrücken: Lambert Academic, 2012. Print.

S. Thomson, C. Huitema, V. Ksinanat, M. Souissi, “DNS Extensions to Support IP Version 6”, RFC 3596, AFNIC, October 2003, <http://tools.ietf.org/html/rfc3596>.

Stevens, W. Richard, Bill Fenner, and Andrew M. Rudoff. UNIX Network Programming The Sockets Networking API. 3rd ed. Vol. 1. Boston: Pearson Education, 2012. Print.

"What Is OSI Model & the Overall Explanation of ISO 7 Layers." Cisco. 27 Mar. 2013. Web. <http://www.cisco1900router.com/what-is-ios-model-the-overall-explanation-of-ios-7-layers.html>. <http://www.cisco1900router.com/wp-content/uploads/2013/03/1-tutorial-osi-7-layer-model.gif>.