dnsop J. Dickinson
Internet-Draft J. Hague
Intended status: Standards Track S. Dickinson
Expires: July 7, 2018 Sinodun IT
T. Manderson
J. Bond
ICANN
January 3, 2018

C-DNS: A DNS Packet Capture Format
draft-ietf-dnsop-dns-capture-format-04

Abstract

This document describes a data representation for collections of DNS messages. The format is designed for efficient storage and transmission of large packet captures of DNS traffic; it attempts to minimize the size of such packet capture files but retain the full DNS message contents along with the most useful transport metadata. It is intended to assist with the development of DNS traffic monitoring applications.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 7, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

There has long been a need to collect DNS queries and responses on authoritative and recursive name servers for monitoring and analysis. This data is used in a number of ways including traffic monitoring, analyzing network attacks and "day in the life" (DITL) [ditl] analysis.

A wide variety of tools already exist that facilitate the collection of DNS traffic data, such as DSC [dsc], packetq [packetq], dnscap [dnscap] and dnstap [dnstap]. However, there is no standard exchange format for large DNS packet captures. The PCAP [pcap] or PCAP-NG [pcapng] formats are typically used in practice for packet captures, but these file formats can contain a great deal of additional information that is not directly pertinent to DNS traffic analysis and thus unnecessarily increases the capture file size.

There has also been work on using text based formats to describe DNS packets such as [I-D.daley-dnsxml], [I-D.hoffman-dns-in-json], but these are largely aimed at producing convenient representations of single messages.

Many DNS operators may receive hundreds of thousands of queries per second on a single name server instance so a mechanism to minimize the storage size (and therefore upload overhead) of the data collected is highly desirable.

The format described in this document, C-DNS (Compacted-DNS), focusses on the problem of capturing and storing large packet capture files of DNS traffic. with the following goals in mind:

- Minimize the file size for storage and transmission

- Minimizing the overhead of producing the packet capture file and the cost of any further (general purpose) compression of the file

This document contains:

- A discussion of the some common use cases in which such DNS data is collected Section 3

- A discussion of the major design considerations in developing an efficient data representation for collections of DNS messages Section 4

- A conceptual overview of the C-DNS format Section 5

- A description of why CBOR [RFC7049] was chosen for this format Section 6

- The definition of the C-DNS format for the collection of DNS messages Section 7.

- Notes on converting C-DNS data to PCAP format Section 9

- Some high level implementation considerations for applications designed to produce C-DNS Section 10

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

"Packet" refers to individual IPv4 or IPv6 packets. Typically these are UDP, but may be constructed from a TCP packet. "Message", unless otherwise qualified, refers to a DNS payload extracted from a UDP or TCP data stream.

The parts of DNS messages are named as they are in [RFC1035]. In specific, the DNS message has five sections: Header, Question, Answer, Authority, and Additional.

Pairs of DNS messages are called a Query and a Response.

3. Data Collection Use Cases

In an ideal world, it would be optimal to collect full packet captures of all packets going in or out of a name server. However, there are several design choices or other limitations that are common to many DNS installations and operators.

o DNS servers are hosted in a variety of situations

  * Self-hosted servers

  * Third party hosting (including multiple third parties)

  * Third party hardware (including multiple third parties)

o Data is collected under different conditions

  * On well-provisioned servers running in a steady state

  * On heavily loaded servers

  * On virtualized servers

  * On servers that are under DoS attack

  * On servers that are unwitting intermediaries in DoS attacks

o Traffic can be collected via a variety of mechanisms

  * On the same hardware as the name server itself

  * Using a network tap on an adjacent host to listen to DNS traffic

  * Using port mirroring to listen from another host

o The capabilities of data collection (and upload) networks vary

  * Out-of-band networks with the same capacity as the in-band network

  - Out-of-band networks with less capacity than the in-band network

  - Everything being on the in-band network

Thus, there is a wide range of use cases from very limited data collection environments (third party hardware, servers that are under attack, packet capture on the name server itself and no out-of-band network) to "limitless" environments (self hosted, well provisioned servers, using a network tap or port mirroring with an out-of-band networks with the same capacity as the in-band network). In the former, it is infeasible to reliably collect full packet captures, especially if the server is under attack. In the latter case, collection of full packet captures may be reasonable.

As a result of these restrictions, the C-DNS data format was designed with the most limited use case in mind such that:

- data collection will occur on the same hardware as the name server itself

- collected data will be stored on the same hardware as the name server itself, at least temporarily

- collected data being returned to some central analysis system will use the same network interface as the DNS queries and responses

- there can be multiple third party servers involved

Because of these considerations, a major factor in the design of the format is minimal storage size of the capture files.

Another significant consideration for any application that records DNS traffic is that the running of the name server software and the transmission of DNS queries and responses are the most important jobs of a name server; capturing data is not. Any data collection system co-located with the name server needs to be intelligent enough to carefully manage its CPU, disk, memory and network utilization. This leads to designing a format that requires a relatively low overhead to produce and minimizes the requirement for further potentially costly compression.

However, it was also essential that interoperability with less restricted infrastructure was maintained. In particular, it is highly desirable that the collection format should facilitate the re-creation of common formats (such as PCAP) that are as close to the original as is realistic given the restrictions above.

4. Design Considerations

This section presents some of the major design considerations used in the development of the C-DNS format.

1. The basic unit of data is a combined DNS Query and the associated Response (a "Q/R data item"). The same structure will be used for unmatched Queries and Responses. Queries without Responses will be captured omitting the response data. Responses without queries will be captured omitting the Query data (but using the Question section from the response, if present, as an identifying QNAME).

   * Rationale: A Query and Response represents the basic level of a clients interaction with the server. Also, combining the Query and Response into one item often reduces storage requirements due to commonality in the data of the two messages.

2. Each Q/R data item will comprise a default Q/R data description and a set of optional sections. Inclusion of optional sections shall be configurable.

   * Rationale: Different users will have different requirements for data to be available for analysis. Users with minimal requirements should not have to pay the cost of recording full data, however this will limit the ability to reconstruct packet captures. For example, omitting the resource records from a Response will reduce the files size, and in principle responses can be synthesized if there is enough context.

3. Multiple Q/R data items will be collected into blocks in the format. Common data in a block will be abstracted and referenced from individual Q/R data items by indexing. The maximum number of Q/R data items in a block will be configurable.

   * Rationale: This blocking and indexing provides a significant reduction in the volume of file data generated. Although this introduces complexity, it provides compression of the data that makes use of knowledge of the DNS message structure.

   * It is anticipated that the files produced can be subject to further compression using general purpose compression tools. Measurements show that blocking significantly reduces the CPU required to perform such strong compression. See Appendix C.2.

   - [TODO: Further discussion of commonality between DNS messages e.g. common query signatures, a finite set of valid responses from authoritatives]

4. Metadata about other packets received can optionally be included in each block. For example, counts of malformed DNS packets and non-DNS packets (e.g. ICMP, TCP resets) sent to the server may be of interest.

5. The wire format content of malformed DNS packets can optionally be recorded.

   - Rationale: Any structured capture format that does not capture the DNS payload byte for byte will be limited to some extent in that it cannot represent "malformed" DNS packets (see Section 8). Only those packets that can be transformed reasonably into the structured format can be represented by the format. However this can result in rather misleading statistics. For example, a malformed query which cannot be represented in the C-DNS format will lead to the (well formed) DNS responses with error code FORMERR appearing as 'unmatched'. Therefore it can greatly aid downstream analysis to have the wire format of the malformed DNS packets available directly in the C-DNS file.

5. Conceptual Overview

The following figures show purely schematic representations of the C-DNS format to convey the high-level structure of the C-DNS format. Section 7 provides a detailed discussion of the CBOR representation and individual elements.

Figure showing the C-DNS format (PNG) [1]

Figure showing the C-DNS format (SVG) [2]

Figure showing the Q/R data item and Block tables format (PNG) [3]

Figure showing the Q/R data item and Block tables format (SVG) [4]

6. Choice of CBOR

This document presents a detailed format description using CBOR, the Concise Binary Object Representation defined in [RFC7049].

The choice of CBOR was made taking a number of factors into account.

- CBOR is a binary representation, and thus is economical in storage space.

- Other binary representations were investigated, and whilst all had attractive features, none had a significant advantage over CBOR. See Appendix C for some discussion of this.

- CBOR is an IETF standard and familiar to IETF participants. It is based on the now-common ideas of lists and objects, and thus requires very little familiarization for those in the wider industry.

- CBOR is a simple format, and can easily be implemented from scratch if necessary. More complex formats require library support which may present problems on unusual platforms.

- CBOR can also be easily converted to text formats such as JSON ([RFC7159]) for debugging and other human inspection requirements.

- CBOR data schemas can be described using CDDL [I-D.greevenbosch-appsawg-cbor-cddl].

7. The C-DNS format

7.1. CDDL definition

The CDDL definition for the C-DNS format is given in Appendix A.

7.2. Format overview

A C-DNS file begins with a file header containing a file type identifier and a preamble. The preamble contains information on the collection settings.

The file header is followed by a series of data blocks.

A block consists of a block header, containing various tables of common data, and some statistics for the traffic received over the block. The block header is then followed by a list of the Q/R data items detailing the queries and responses received during processing of the block input. The list of Q/R data items is in turn followed by a list of per-client counts of particular IP events that occurred during collection of the block data.

The exact nature of the DNS data will affect what block size is the best fit, however sample data for a root server indicated that block sizes up to 10,000 Q/R data items give good results. See Appendix C.6 for more details.

If no field type is specified, then the field is unsigned.

In all quantities that contain bit flags, bit 0 indicates the least significant bit. An item described as an index is the index of the Q/R data item in the referenced table. Indexes are 1-based. An index value of 0 is reserved to mean "not present".

All map keys are unsigned integers with values specified in the CDDL (string keys would significantly bloat the file size).

7.3. File header contents

The file header contains the following:

Field	Type	Description
file-type-id	Text string	String "C-DNS" identifying the file type.
file-preamble	Map of items	Collection information for the whole file.
file-blocks	Array of Blocks	The data blocks.

7.4. File preamble contents

The file preamble contains the following:

Field	Type	Description
major-format-version	Unsigned	Unsigned integer '1'. The major version of format used in file.
minor-format-version	Unsigned	Unsigned integer '0'. The minor version of format used in file.
private-version	Unsigned	Version indicator available for private use by applications. Optional.
configuration	Map of items	The collection configuration. Optional.
generator-id	Text string	String identifying the collection program. Optional.
host-id	Text string	String identifying the collecting host. Empty if converting an existing packet capture file. Optional.

7.5. Configuration contents

The collection configuration contains the following items. All are optional.

Field	Type	Description
query-timeout	Unsigned	To be matched with a query, a response must arrive within this number of seconds.
skew-timeout	Unsigned	The network stack may report a response before the corresponding query. A response is not considered to be missing a query until after this many micro-seconds.
snaplen	Unsigned	Collect up to this many bytes per packet.
promisc	Unsigned	1 if promiscuous mode was enabled

		on the interface, 0 otherwise.
interfaces	Array of text strings	Identifiers of the interfaces used for collection.
server-addresses	Array of byte strings	Server collection IP addresses. Hint for downstream analysers; does not affect collection.
vlan-ids	Array of unsigned	Identifiers of VLANs selected for collection.
filter	Text string	'tcpdump' [pcap] style filter for input.
query-options	Unsigned	Bit flags indicating sections in Query messages to be collected. Bit 0. Collect second and subsequent Questions in the Question section. Bit 1. Collect Answer sections. Bit 2. Collect Authority sections. Bit 3. Collection Additional sections.
response-options	Unsigned	Bit flags indicating sections in Response messages to be collected. Bit 0. Collect second and subsequent Questions in the Question section. Bit 1. Collect Answer sections. Bit 2. Collect Authority sections. Bit 3. Collection Additional sections.
accept-rr-types	Array of text strings	A set of RR type names [rrtypes]. If not empty, only the nominated RR types are collected.
ignore-rr-types	Array of text strings	A set of RR type names [rrtypes]. If not empty, all RR types are collected except those listed. If present, this item must be empty if a non-empty list of Accept RR

		types is present.
max-block-qr-items	Unsigned	Maximum number of Q/R data items in a block.
collect-malformed	Unsigned	1 if malformed packet contents are collected, 0 otherwise.

7.6. Block contents

Each block contains the following:

Field	Type	Description
preamble	Map of items	Overall information for the block.
statistics	Map of statistics	Statistics about the block. Optional.
tables	Map of tables	The tables containing data referenced by individual Q/R data items.
queries	Array of Q/R data items	Details of individual Q/R data items.
address-event-counts	Array of Address Event counts	Per client counts of ICMP messages and TCP resets. Optional.
malformed-packet-data	Array of malformed packets	Wire contents of malformed packets. Optional.

7.7. Block preamble map

The block preamble map contains overall information for the block.

Field	Type	Description
earliest-time	Array of unsigned	A timestamp for the earliest record in the block. The timestamp is specified as a CBOR array with two or three elements. The first two elements are as in Posix struct timeval. The first element is an unsigned integer time_t and the second is an unsigned integer number of microseconds. The third, if present, is an unsigned integer number of picoseconds. The microsecond and picosecond items always have a value between 0 and 999,999.

7.8. Block statistics

The block statistics section contains some basic statistical information about the block. All are optional.

Field	Type	Description
total-packets	Unsigned	Total number of packets processed from the input traffic stream during collection of the block data.
total-pairs	Unsigned	Total number of Q/R data items in the block.
unmatched-queries	Unsigned	Number of unmatched queries in the block.
unmatched-responses	Unsigned	Number of unmatched responses in the block.
malformed-packets	Unsigned	Number of malformed packets found in input for the block.

Implementations may choose to add additional implementation-specific fields to the statistics.

7.9. Block table map

The block table map contains the block tables. Each element, or table, is an array. The following tables detail the contents of each block table.

The Present column in the following tables indicates the circumstances when an optional field will be present. A Q/R data item may be:

- A Query plus a Response.
- A Query without a Response.
- A Response without a Query.

Also:

- A Query and/or a Response may contain an OPT section.
- A Question may or may not be present. If the Query is available, the Question section of the Query is used. If no Query is available, the Question section of the Response is used. Unless otherwise noted, a Question refers to the first Question in the Question section.

So, for example, a field listed with a Present value of QUERY is present whenever the Q/R data item contains a Query. If the pair contains a Response only, the field will not be present.

7.10. IP address table

The table "ip-address" holds all client and server IP addresses in the block. Each item in the table is a single IP address.

Field	Type	Description
ip-address	Byte string	The IP address, in network byte order. The string is 4 bytes long for an IPv4 address, 16 bytes long for an IPv6 address.

7.11. Class/Type table

The table "classtype" holds pairs of RR CLASS and TYPE values. Each item in the table is a CBOR map.

Field	Type	Description
type	Unsigned	TYPE value.
class	Unsigned	CLASS value.

7.12. Name/RDATA table

The table "name-rdata" holds the contents of all NAME or RDATA items in the block. Each item in the table is the content of a single NAME or RDATA.

Note that NAMEs, and labels within RDATA contents, are full domain names or labels; no DNS style name compression is used on the individual names/labels within the format.

Field	Type	Description
name-rdata	Byte string	The NAME or RDATA contents (uncompressed).

7.13. Query Signature table

The table "query-sig" holds elements of the Q/R data item that are often common between multiple individual Q/R data items. Each item in the table is a CBOR map. Each item in the map has an unsigned value and an unsigned integer key.

The following abbreviations are used in the Present (P) column

- Q = QUERY
- A = Always
- QT = QUESTION
- QO = QUERY, OPT
- QR = QUERY & RESPONSE
- R = RESPONSE

Field	P	Description

server-address-index	A	The index in the IP address table of the server IP address.
server-port	A	The server port.
transport-flags	A	Bit flags describing the transport used to service the query. Bit 0 is the least significant bit. Bit 0. Transport type. 0 = UDP, 1 = TCP. Bit 1. IP type. 0 = IPv4, 1 = IPv6. Bit 2. Trailing bytes in query payload. The DNS query message in the UDP payload was followed by some additional bytes, which were discarded.
qr-sig-flags	A	Bit flags indicating information present in this Q/R data item. Bit 0 is the least significant bit. Bit 0. 1 if a Query is present. Bit 1. 1 if a Response is present. Bit 2. 1 if one or more Question is present. Bit 3. 1 if a Query is present and it has an OPT Resource Record. Bit 4. 1 if a Response is present and it has an OPT Resource Record. Bit 5. 1 if a Response is present but has no Question.
query-opcode	Q	Query OPCODE. Optional.
qr-dns-flags	A	Bit flags with values from the Query and Response DNS flags. Bit 0 is the least significant bit. Flag values are 0 if the Query or Response is not present. Bit 0. Query Checking Disabled (CD). Bit 1. Query Authenticated Data (AD). Bit 2. Query reserved (Z). Bit 3. Query Recursion Available (RA). Bit 4. Query Recursion Desired (RD). Bit 5. Query TrunCation (TC). Bit 6. Query Authoritative Answer

		(AA). Bit 7. Query DNSSEC answer OK (DO). Bit 8. Response Checking Disabled (CD). Bit 9. Response Authenticated Data (AD). Bit 10. Response reserved (Z). Bit 11. Response Recursion Available (RA). Bit 12. Response Recursion Desired (RD). Bit 13. Response TrunCation (TC). Bit 14. Response Authoritative Answer (AA).
query-rcode	Q	Query RCODE. If the Query contains OPT, this value incorporates any EXTENDED_RCODE_VALUE. Optional.
query-classtype-index	QT	The index in the Class/Type table of the CLASS and TYPE of the first Question. Optional.
query-qd-count	QT	The QDCOUNT in the Query, or Response if no Query present. Optional.
query-an-count	Q	Query ANCOUNT. Optional.
query-ar-count	Q	Query ARCOUNT. Optional.
query-ns-count	Q	Query NSCOUNT. Optional.
edns-version	QO	The Query EDNS version. Optional.
udp-buf-size	QO	The Query EDNS sender's UDP payload size. Optional.
opt-rdata-index	QO	The index in the NAME/RDATA table of the OPT RDATA. Optional.
response-rcode	R	Response RCODE. If the Response contains OPT, this value incorporates any EXTENDED_RCODE_VALUE. Optional.

7.14. Question table

The table "qrr" holds details on individual Questions in a Question section. Each item in the table is a CBOR map containing a single Question. Each item in the map has an unsigned value and an unsigned integer key. This data is optionally collected.

Field	Description
name-index	The index in the NAME/RDATA table of the QNAME.
classtype-index	The index in the Class/Type table of the CLASS and TYPE of the Question.

7.15. Resource Record (RR) table

The table "rr" holds details on individual Resource Records in RR sections. Each item in the table is a CBOR map containing a single Resource Record. This data is optionally collected.

Field	Description
name-index	The index in the NAME/RDATA table of the NAME.
classtype-index	The index in the Class/Type table of the CLASS and TYPE of the RR.
ttl	The RR Time to Live.
rdata-index	The index in the NAME/RDATA table of the RR RDATA.

7.16. Question list table

The table "qlist" holds a list of second and subsequent individual Questions in a Question section. Each item in the table is a CBOR unsigned integer. This data is optionally collected.

Field	Description
question	The index in the Question table of the individual Question.

7.17. Resource Record list table

The table "rrlist" holds a list of individual Resource Records in a Answer, Authority or Additional section. Each item in the table is a CBOR unsigned integer. This data is optionally collected.

Field	Description
rr	The index in the Resource Record table of the individual Resource Record.

7.18. Query/Response data

The block Q/R data is a CBOR array of individual Q/R data items. Each item in the array is a CBOR map containing details on the individual Q/R data item.

Note that there is no requirement that the elements of the Q/R array are presented in strict chronological order.

The following abbreviations are used in the Present (P) column

- Q = QUERY
- A = Always
- QT = QUESTION
- QO = QUERY, OPT
- QR = QUERY & RESPONSE
- R = RESPONSE

Each item in the map has an unsigned value (with the exception of those listed below) and an unsigned integer key.

- query-extended and response-extended which are of type Extended Information.
- delay-useconds and delay-pseconds which are integers (The delay can be negative if the network stack/capture library returns them out of order.)

Field	P	Description

time-useconds	A	Q/R timestamp as an offset in microseconds from the Block preamble Timestamp. The timestamp is the timestamp of the Query, or the Response if there is no Query.
time-pseconds	A	Picosecond component of the timestamp. Optional.
client-address-index	A	The index in the IP address table of the client IP address.
client-port	A	The client port.
transaction-id	A	DNS transaction identifier.
query-signature-index	A	The index of the Query Signature table record for this data item.
client-hoplimit	Q	The IPv4 TTL or IPv6 Hoplimit from the Query packet. Optional.
delay-useconds	QR	The time difference between Query and Response, in microseconds. Only present if there is a query and a response.
delay-pseconds	QR	Picosecond component of the time different between Query and Response. If delay-useconds is non-zero then delay-pseconds (if present) MUST be of the same sign as delay-useconds, or be 0. Optional.
query-name-index	QT	The index in the NAME/RDATA table of the QNAME for the first Question. Optional.
query-size	R	DNS query message size (see below). Optional.
response-size	R	DNS query message size (see below). Optional.
query-extended	Q	Extended Query information. This item is only present if collection of extra Query information is

		configured. Optional.
response-extended	R	Extended Response information. This item is only present if collection of extra Response information is configured. Optional.

An implementation must always collect basic Q/R information. It may be configured to collect details on Question, Answer, Authority and Additional sections of the Query, the Response or both. Note that only the second and subsequent Questions of any Question section are collected (the details of the first are in the basic information), and that OPT Records are not collected in the Additional section.

The query-size and response-size fields hold the DNS message size. For UDP this is the size of the UDP payload that contained the DNS message and will therefore include any trailing bytes if present. Trailing bytes with queries are routinely observed in traffic to authoritative servers and this value allows a calculation of how many trailing bytes were present. For TCP it is the size of the DNS message as specified in the two-byte message length header.

The Extended information is a CBOR map as follows. Each item in the map is present only if collection of the relevant details is configured. Each item in the map has an unsigned value and an unsigned integer key.

Field	Description
question-index	The index in the Questions list table of the entry listing any second and subsequent Questions in the Question section for the Query or Response.
answer-index	The index in the RR list table of the entry listing the Answer Resource Record sections for the Query or Response.
authority-index	The index in the RR list table of the entry listing the Authority Resource Record sections for the Query or Response.
additional-index	The index in the RR list table of the entry listing the Additional Resource Record sections for the Query or Response.

7.19. Address Event counts

This table holds counts of various IP related events relating to traffic with individual client addresses.

Field	Type	Description
ae-type	Unsigned	The type of event. The following events types are currently defined: 0. TCP reset. 1. ICMP time exceeded. 2. ICMP destination unreachable. 3. ICMPv6 time exceeded. 4. ICMPv6 destination unreachable. 5. ICMPv6 packet too big.
ae-code	Unsigned	A code relating to the event. Optional.
ae-address-index	Unsigned	The index in the IP address table of the client address.
ae-count	Unsigned	The number of occurrences of this event during the block collection period.

7.20. Malformed packet records

This optional table records the original wire format content of malformed packets (see Section 8).

Field	Type	Description
time-useconds	A	Packet timestamp as an offset in microseconds from the Block preamble Timestamp.
time-pseconds	A	Picosecond component of the timestamp. Optional.
packet-content	Byte string	The packet content in wire format.

8. Malformed Packets

In the context of generating a C-DNS file it is assumed that only those packets which can be parsed to produce a well-formed DNS message are stored in the C-DNS format. This means as a minimum:

- The packet has a well-formed 12 bytes DNS Header

- The section counts are consistent with the section contents

- All of the resource records can be parsed

In principle, packets that do not meet these criteria could be classified into two categories:

- Partially malformed: those packets which can be decoded sufficiently to extract

  * a DNS header (and therefore a DNS transaction ID)

  * a QDCOUNT

  * the first Question in the Question section if QDCOUNT is greater than 0

  but suffer other issues while parsing. This is the minimum information required to attempt Query/Response matching as described in Section 10.1

- Completely malformed: those packets that cannot be decoded to this extent.

An open question is whether there is value in attempting to process partially malformed packets in an analogous manner to well formed packets in terms of attempting to match them with the corresponding query or response. This could be done by creating 'placeholder' records during Query/Response matching with just the information extracted as above. If the packet were then matched the resulting C-DNS Q/R data item would include a flag to indicate a malformed record (in addition to capturing the wire format of the packet).

An advantage of this would be that it would result in more meaningful statistics about matched packets because, for example, some partially malformed queries could be matched to responses. However it would only apply to those queries where the first Question is well formed. It could also simplify the downstream analysis of C-DNS files and the reconstruction of packet streams from C-DNS.

A disadvantage is that this adds complexity to the Query/Response matching and data representation, could potentially lead to false matches and some additional statistics would be required (e.g. counts for matched-partially-malformed, unmatched-partially-malformed, completely-malformed).

9. C-DNS to PCAP

It is possible to re-construct PCAP files from the C-DNS format in a lossy fashion. Some of the issues with reconstructing both the DNS payload and the full packet stream are outlined here.

The reconstruction depends on whether or not all the optional sections of both the query and response were captured in the C-DNS file. Clearly, if they were not all captured, the reconstruction will be imperfect.

Even if all sections of the response were captured, one cannot reconstruct the DNS response payload exactly due to the fact that some DNS names in the message on the wire may have been compressed. Section 9.1 discusses this is more detail.

Some transport information is not captured in the C-DNS format. For example, the following aspects of the original packet stream cannot be re-constructed from the C-DNS format:

- IP fragmentation

- TCP stream information:

  - Multiple DNS messages may have been sent in a single TCP segment

  - A DNS payload may have be split across multiple TCP segments

  - Multiple DNS messages may have be sent on a single TCP session

- Malformed DNS messages if the wire format is not recorded

- Any Non-DNS messages that were in the original packet stream e.g. ICMP

Simple assumptions can be made on the reconstruction: fragmented and DNS-over-TCP messages can be reconstructed into single packets and a single TCP session can be constructed for each TCP packet.

Additionally, if malformed packets and Non-DNS packets are captured separately, they can be merged with packet captures reconstructed from C-DNS to produce a more complete packet stream.

9.1. Name Compression

All the names stored in the C-DNS format are full domain names; no DNS style name compression is used on the individual names within the format. Therefore when reconstructing a packet, name compression must be used in order to reproduce the on the wire representation of the packet.

[RFC1035] name compression works by substituting trailing sections of a name with a reference back to the occurrence of those sections earlier in the message. Not all name server software uses the same algorithm when compressing domain names within the responses. Some attempt maximum recompression at the expense of runtime resources, others use heuristics to balance compression and speed and others use different rules for what is a valid compression target.

This means that responses to the same question from different name server software which match in terms of DNS payload content (header, counts, RRs with name compression removed) do not necessarily match byte-for-byte on the wire.

Therefore, it is not possible to ensure that the DNS response payload is reconstructed byte-for-byte from C-DNS data. However, it can at least, in principle, be reconstructed to have the correct payload length (since the original response length is captured) if there is enough knowledge of the commonly implemented name compression algorithms. For example, a simplistic approach would be to try each algorithm in turn to see if it reproduces the original length, stopping at the first match. This would not guarantee the correct algorithm has been used as it is possible to match the length whilst still not matching the on the wire bytes but, without further information added to the C-DNS data, this is the best that can be achieved.

Appendix B presents an example of two different compression algorithms used by well-known name server software.

10. Data Collection

This section describes a non-normative proposed algorithm for the processing of a captured stream of DNS queries and responses and matching queries/responses where possible.

For the purposes of this discussion, it is assumed that the input has been pre-processed such that:

1. All IP fragmentation reassembly, TCP stream reassembly, and so on, has already been performed

2. Each message is associated with transport metadata required to generate the Primary ID (see Section 10.2.1)

3. Each message has a well-formed DNS header of 12 bytes and (if present) the first Question in the Question section can be parsed to generate the Secondary ID (see below). As noted earlier, this requirement can result in a malformed query being removed in the pre-processing stage, but the correctly formed response with RCODE of FORMERR being present.

DNS messages are processed in the order they are delivered to the application. It should be noted that packet capture libraries do not necessary provide packets in strict chronological order.

TODO: Discuss the corner cases resulting from this in more detail.

10.1. Matching algorithm

A schematic representation of the algorithm for matching Q/R data items is shown in the following diagram:

Figure showing the Query/Response matching algorithm format (PNG) [5]

Figure showing the Query/Response matching algorithm format (SVG) [6]

Further details of the algorithm are given in the following sections.

10.2. Message identifiers

10.2.1. Primary ID (required)

A Primary ID is constructed for each message. It is composed of the following data:

1. Source IP Address

2. Destination IP Address

3. Source Port

4. Destination Port

5. Transport

6. DNS Message ID

10.2.2. Secondary ID (optional)

If present, the first Question in the Question section is used as a secondary ID for each message. Note that there may be well formed DNS queries that have a QDCOUNT of 0, and some responses may have a QDCOUNT of 0 (for example, responses with RCODE=FORMERR or NOTIMP). In this case the secondary ID is not used in matching.

10.3. Algorithm Parameters

1. Query timeout

2. Skew timeout

10.4. Algorithm Requirements

The algorithm is designed to handle the following input data:

1. Multiple queries with the same Primary ID (but different Secondary ID) arriving before any responses for these queries are seen.

2. Multiple queries with the same Primary and Secondary ID arriving before any responses for these queries are seen.

3. Queries for which no later response can be found within the specified timeout.

4. Responses for which no previous query can be found within the specified timeout.

10.5. Algorithm Limitations

For cases 1 and 2 listed in the above requirements, it is not possible to unambiguously match queries with responses. This algorithm chooses to match to the earliest query with the correct Primary and Secondary ID.

10.6. Workspace

A FIFO structure is used to hold the Q/R data items during processing.

10.7. Output

The output is a list of Q/R data items. Both the Query and Response elements are optional in these items, therefore Q/R data items have one of three types of content:

1. A matched pair of query and response messages

2. A query message with no response

3. A response message with no query

The timestamp of a list item is that of the query for cases 1 and 2 and that of the response for case 3.

10.8. Post Processing

When ending capture, all remaining entries in the Q/R data item FIFO should be treated as timed out queries.

11. Implementation Status

[Note to RFC Editor: please remove this section and reference to [RFC7942] prior to publication.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

11.1. DNS-STATS Compactor

ICANN/Sinodun IT have developed an open source implementation called DNS-STATS Compactor. The Compactor is a suite of tools which can capture DNS traffic (from either a network interface or a PCAP file) and store it in the Compacted-DNS (C-DNS) file format. PCAP files for the captured traffic can also be reconstructed. See Compactor [7].

This implementation:

- is mature but has only been deployed for testing in a single environment so is not yet classified as production ready.

- covers the whole of the specification described in the -03 draft with the exception of support for malformed packets (Section 8) and pico second time resolution. (Note: this implementation does allow malformed packets to be dumped to a PCAP file).

- is released under the Mozilla Public License Version 2.0.

- has a users mailing list available, see dns-stats-users [8].

There is also some discussion of issues encountered during development available at Compressing Pcap Files [9] and Packet Capture [10].

This information was last updated on 29th of June 2017.

12. IANA Considerations

None

13. Security Considerations

Any control interface MUST perform authentication and encryption.

Any data upload MUST be authenticated and encrypted.

14. Acknowledgements

The authors wish to thank CZ.NIC, in particular Tomas Gavenciak, for many useful discussions on binary formats, compression and packet matching. Also Jan Vcelak and Wouter Wijngaards for discussions on name compression and Paul Hoffman for a detailed review of the document and the C-DNS CDDL.

Thanks also to Robert Edmonds and Jerry Lundstroem for review.

Also, Miek Gieben for mmark [11]

15. Changelog

draft-ietf-dnsop-dns-capture-format-04

- Correct query-d0 to query-do in CDDL
- Clarify that map keys are unsigned integers
- Add Type to Class/type table
- Clarify storage format in section 7.12

draft-ietf-dnsop-dns-capture-format-03

- Added an Implementation Status section

draft-ietf-dnsop-dns-capture-format-02

- Update qr_data_format.png to match CDDL
- Editorial clarifications and improvements

draft-ietf-dnsop-dns-capture-format-01

- Many editorial improvements by Paul Hoffman
- Included discussion of malformed packet handling
- Improved Appendix C on Comparison of Binary Formats
- Now using C-DNS field names in the tables in section 8
- A handful of new fields included (CDDL updated)
- Timestamps now include optional picoseconds
- Added details of block statistics

draft-ietf-dnsop-dns-capture-format-00

- Changed dnstap.io to dnstap.info
- qr_data_format.png was cut off at the bottom
- Update authors address

- Improve wording in Abstract

- Changed DNS-STAT to C-DNS in CDDL

- Set the format version in the CDDL

- Added a TODO: Add block statistics

- Added a TODO: Add extend to support pico/nano. Also do this for Time offset and Response delay

- Added a TODO: Need to develop optional representation of malformed packets within C-DNS and what this means for packet matching. This may influence which fields are optional in the rest of the representation.

- Added section on design goals to Introduction

- Added a TODO: Can Class be optimised? Should a class of IN be inferred if not present?

draft-dickinson-dnsop-dns-capture-format-00

- Initial commit

16. References

16.1. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

16.2. Informative References

[ditl] DNS-OARC, "DITL", 2016, <https://www.dns-oarc.net/oarc/data/ditl>.

[dnscap] DNS-OARC, "DNSCAP", 2016, <https://www.dns-oarc.net/tools/dnscap>.

[dnstap] dnstap.info, "dnstap", 2016, <http://dnstap.info/>.

[dsc] Wessels, D. and J. Lundstrom, "DSC", 2016, <https://www.dns-oarc.net/tools/dsc>.

[I-D.daley-dnsxml] Daley, J., Morris, S., and J. Dickinson, "dnsxml - A standard XML representation of DNS data", draft-daley-dnsxml-00 (work in progress), July 2013.

[I-D.greevenbosch-appsawg-cbor-cddl] Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-11 (work in progress), July 2017.

[I-D.hoffman-dns-in-json] Hoffman, P., "Representing DNS Messages in JSON", draft-hoffman-dns-in-json-13 (work in progress), October 2017.

[packetq] .SE - The Internet Infrastructure Foundation, "PacketQ", 2014, <https://github.com/dotse/PacketQ>.

[pcap] tcpdump.org, "PCAP", 2016, <http://www.tcpdump.org/>.

[pcapng] Tuexen, M., Risso, F., Bongertz, J., Combs, G., and G. Harris, "pcap-ng", 2016, <https://github.com/pcapng/pcapng>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[rrtypes] IANA, "RR types", 2016, <http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml#dns-parameters-4>.

16.3. URIs

[1] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-04/cdns_format.png

[2] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-04/cdns_format.svg

[3] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-04/qr_data_format.png

[4] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-04/qr_data_format.svg

[5] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-04/packet_matching.png

[6] https://github.com/dns-stats/draft-dns-capture-format/blob/master/draft-04/packet_matching.svg

[7] https://github.com/dns-stats/compactor/wiki

[8] https://mm.dns-stats.org/mailman/listinfo/dns-stats-users

[9] https://www.sinodun.com/2017/06/compressing-pcap-files/

[10] https://www.sinodun.com/2017/06/more-on-debian-jessieubuntu-trusty-packet-capture-woes/

[11] https://github.com/miekg/mmark

[12] https://www.nlnetlabs.nl/projects/nsd/

[13] https://www.knot-dns.cz/

[14] https://avro.apache.org/

[15] https://developers.google.com/protocol-buffers/

[16] http://cbor.io

[17] https://github.com/kubo/snzip

[18] http://google.github.io/snappy/

[19] http://lz4.github.io/lz4/

[20] http://www.gzip.org/

[21] http://facebook.github.io/zstd/

[22] http://tukaani.org/xz/

[23] https://github.com/dns-stats/draft-dns-capture-format/blob/master/file-size-versus-block-size.png

[24] https://github.com/dns-stats/draft-dns-capture-format/blob/master/file-size-versus-block-size.svg

Appendix A. CDDL

```
; CDDL specification of the file format for C-DNS,
; which describes a collection of DNS messages and
; traffic meta-data.

File = [
    file-type-id  : tstr, ; = "C-DNS"
    file-preamble : FilePreamble,
    file-blocks   : [* Block],
]

FilePreamble = {
    major-format-version => uint,  ; = 1
    minor-format-version => uint,  ; = 0
    ? private-version    => uint,
    ? configuration      => Configuration,
    ? generator-id       => tstr,
    ? host-id            => tstr,
}

major-format-version = 0
minor-format-version = 1
private-version      = 2
configuration        = 3
generator-id         = 4
host-id              = 5

Configuration = {
    ? query-timeout      => uint,
    ? skew-timeout       => uint,
    ? snaplen            => uint,
    ? promisc            => uint,
    ? interfaces         => [* tstr],
    ? server-addresses   => [* IPAddress], ; Hint for later analysis
    ? vlan-ids           => [* uint],
    ? filter             => tstr,
    ? query-options      => QRCollectionSections,
```

```
      ? response-options   => QRCollectionSections,
      ? accept-rr-types    => [* uint],
      ? ignore-rr-types    => [* uint],
      ? max-block-qr-items => uint,
      ? collect-malformed  => uint,
  }

  QRCollectionSectionValues = &(
      question  : 0, ; Second & subsequent questions
      answer    : 1,
      authority : 2,
      additional: 3,
  )
  QRCollectionSections = uint .bits QRCollectionSectionValues

  query-timeout      = 0
  skew-timeout       = 1
  snaplen            = 2
  promisc            = 3
  interfaces         = 4
  vlan-ids           = 5
  filter             = 6
  query-options      = 7
  response-options   = 8
  accept-rr-types    = 9
  ignore-rr-types    = 10
  server-addresses   = 11
  max-block-qr-items = 12
  collect-malformed  = 13

  Block = {
      preamble                => BlockPreamble,
      ? statistics            => BlockStatistics,
      tables                  => BlockTables,
      queries                 => [* QueryResponse],
      ? address-event-counts  => [* AddressEventCount],
      ? malformed-packet-data => [* MalformedPacket],
  }

  preamble              = 0
  statistics            = 1
  tables                = 2
  queries               = 3
  address-event-counts  = 4
  malformed-packet-data = 5

  BlockPreamble = {
      earliest-time => Timeval
```

```
  }

  earliest-time = 1

  Timeval = [
      seconds       : uint,
      microseconds  : uint,
      ? picoseconds : uint,
  ]

  BlockStatistics = {
      ? total-packets                => uint,
      ? total-pairs                  => uint,
      ? unmatched-queries            => uint,
      ? unmatched-responses          => uint,
      ? malformed-packets            => uint,
  }

  total-packets                = 0
  total-pairs                  = 1
  unmatched-queries            = 2
  unmatched-responses          = 3
  malformed-packets            = 4

  BlockTables = {
      ip-address => [* IPAddress],
      classtype  => [* ClassType],
      name-rdata => [* bstr], ; Holds both Name RDATA and RDATA
      query-sig  => [* QuerySignature]
      ? qlist    => [* QuestionList],
      ? qrr      => [* Question],
      ? rrlist   => [* RRList],
      ? rr       => [* RR],
  }

  ip-address = 0
  classtype  = 1
  name-rdata = 2
  query-sig  = 3
  qlist      = 4
  qrr        = 5
  rrlist     = 6
  rr         = 7

  QueryResponse = {
      time-useconds         => uint, ; Time offset from start of block
      ? time-pseconds       => uint, ; in microseconds and picoseconds
      client-address-index  => uint,
```

```
    client-port           => uint,
    transaction-id        => uint,
    query-signature-index => uint,
    ? client-hoplimit     => uint,
    ? delay-useconds      => int,
    ? delay-pseconds      => int, ; Has same sign as delay-useconds
    ? query-name-index    => uint,
    ? query-size          => uint, ; DNS size of query
    ? response-size       => uint, ; DNS size of response
    ? query-extended      => QueryResponseExtended,
    ? response-extended   => QueryResponseExtended,
}

time-useconds         = 0
time-pseconds         = 1
client-address-index  = 2
client-port           = 3
transaction-id        = 4
query-signature-index = 5
client-hoplimit       = 6
delay-useconds        = 7
delay-pseconds        = 8
query-name-index      = 9
query-size            = 10
response-size         = 11
query-extended        = 12
response-extended     = 13

ClassType = {
    type  => uint,
    class => uint,
}

type  = 0
class = 1

DNSFlagValues = &(
    query-cd   : 0,
    query-ad   : 1,
    query-z    : 2,
    query-ra   : 3,
    query-rd   : 4,
    query-tc   : 5,
    query-aa   : 6,
    query-do   : 7,
    response-cd: 8,
    response-ad: 9,
    response-z : 10,
```

```
        response-ra: 11,
        response-rd: 12,
        response-tc: 13,
        response-aa: 14,
    )
    DNSFlags = uint .bits DNSFlagValues

    QueryResponseFlagValues = &(
        has-query               : 0,
        has-reponse             : 1,
        query-has-question      : 2,
        query-has-opt           : 3,
        response-has-opt        : 4,
        response-has-no-question: 5,
    )
    QueryResponseFlags = uint .bits QueryResponseFlagValues

    TransportFlagValues = &(
        tcp               : 0,
        ipv6              : 1,
        query-trailingdata: 2,
    )
    TransportFlags = uint .bits TransportFlagValues

    QuerySignature = {
        server-address-index    => uint,
        server-port             => uint,
        transport-flags         => TransportFlags,
        qr-sig-flags            => QueryResponseFlags,
        ? query-opcode          => uint,
        qr-dns-flags            => DNSFlags,
        ? query-rcode           => uint,
        ? query-classtype-index => uint,
        ? query-qd-count        => uint,
        ? query-an-count        => uint,
        ? query-ar-count        => uint,
        ? query-ns-count        => uint,
        ? edns-version          => uint,
        ? udp-buf-size          => uint,
        ? opt-rdata-index       => uint,
        ? response-rcode        => uint,
    }

    server-address-index  = 0
    server-port           = 1
    transport-flags       = 2
    qr-sig-flags          = 3
    query-opcode          = 4
```

```
  qr-dns-flags          = 5
  query-rcode           = 6
  query-classtype-index = 7
  query-qd-count        = 8
  query-an-count        = 9
  query-ar-count        = 10
  query-ns-count        = 11
  edns-version          = 12
  udp-buf-size          = 13
  opt-rdata-index       = 14
  response-rcode        = 15

  QuestionList = [
      * uint, ; Index of Question
  ]

  Question = {                 ; Second and subsequent questions
      name-index      => uint, ; Index to a name in the name-rdata table
      classtype-index => uint,
  }

  name-index      = 0
  classtype-index = 1

  RRList = [
      * uint, ; Index of RR
  ]

  RR = {
      name-index      => uint, ; Index to a name in the name-rdata table
      classtype-index => uint,
      ttl             => uint,
      rdata-index     => uint, ; Index to RDATA in the name-rdata table
  }

  ttl         = 2
  rdata-index = 3

  QueryResponseExtended = {
      ? question-index   => uint, ; Index of QuestionList
      ? answer-index     => uint, ; Index of RRList
      ? authority-index  => uint,
      ? additional-index => uint,
  }

  question-index   = 0
  answer-index     = 1
  authority-index  = 2
```

```
  additional-index = 3

  AddressEventCount = {
      ae-type          => &AddressEventType,
      ? ae-code        => uint,
      ae-address-index => uint,
      ae-count         => uint,
  }

  ae-type          = 0
  ae-code          = 1
  ae-address-index = 2
  ae-count         = 3

  AddressEventType = (
      tcp-reset              : 0,
      icmp-time-exceeded     : 1,
      icmp-dest-unreachable  : 2,
      icmpv6-time-exceeded   : 3,
      icmpv6-dest-unreachable: 4,
      icmpv6-packet-too-big  : 5,
  )

  MalformedPacket = {
      time-useconds   => uint, ; Time offset from start of block
      ? time-pseconds => uint, ; in microseconds and picoseconds
      packet-content  => bstr, ; Raw packet contents
  }

  time-useconds    = 0
  time-pseconds    = 1
  packet-content   = 2

  IPv4Address = bstr .size 4
  IPv6Address = bstr .size 16
  IPAddress = IPv4Address / IPv6Address
```

Appendix B. DNS Name compression example

The basic algorithm, which follows the guidance in [RFC1035], is simply to collect each name, and the offset in the packet at which it starts, during packet construction. As each name is added, it is offered to each of the collected names in order of collection, starting from the first name. If labels at the end of the name can be replaced with a reference back to part (or all) of the earlier name, and if the uncompressed part of the name is shorter than any compression already found, the earlier name is noted as the compression target for the name.

The following tables illustrate the process. In an example packet, the first name is example.com.

N	Name	Uncompressed	Compression Target
1	example.com		

The next name added is bar.com. This is matched against example.com. The com part of this can be used as a compression target, with the remaining uncompressed part of the name being bar.

N	Name	Uncompressed	Compression Target
1	example.com		
2	bar.com	bar	1 + offset to com

The third name added is www.bar.com. This is first matched against example.com, and as before this is recorded as a compression target, with the remaining uncompressed part of the name being www.bar. It is then matched against the second name, which again can be a compression target. Because the remaining uncompressed part of the name is www, this is an improved compression, and so it is adopted.

N	Name	Uncompressed	Compression Target
1	example.com		
2	bar.com	bar	1 + offset to com
3	www.bar.com	www	2

As an optimization, if a name is already perfectly compressed (in other words, the uncompressed part of the name is empty), then no further names will be considered for compression.

B.1. NSD compression algorithm

Using the above basic algorithm the packet lengths of responses generated by NSD [12] can be matched almost exactly. At the time of writing, a tiny number (<.01%) of the reconstructed packets had incorrect lengths.

B.2. Knot Authoritative compression algorithm

The Knot Authoritative [13] name server uses different compression behavior, which is the result of internal optimization designed to balance runtime speed with compression size gains. In brief, and omitting complications, Knot Authoritative will only consider the QNAME and names in the immediately preceding RR section in an RRSET as compression targets.

A set of smart heuristics as described below can be implemented to mimic this and while not perfect it produces output nearly, but not quite, as good a match as with NSD. The heuristics are:

1. A match is only perfect if the name is completely compressed AND the TYPE of the section in which the name occurs matches the TYPE of the name used as the compression target.

2. If the name occurs in RDATA:

   * If the compression target name is in a query, then only the first RR in an RRSET can use that name as a compression target.

   * The compression target name MUST be in RDATA.

   * The name section TYPE must match the compression target name section TYPE.

   * The compression target name MUST be in the immediately preceding RR in the RRSET.

Using this algorithm less than 0.1% of the reconstructed packets had incorrect lengths.

B.3. Observed differences

In sample traffic collected on a root name server around 2-4% of responses generated by Knot had different packet lengths to those produced by NSD.

Appendix C. Comparison of Binary Formats

Several binary serialisation formats were considered, and for completeness were also compared to JSON.

- Apache Avro [14]. Data is stored according to a pre-defined schema. The schema itself is always included in the data file.

Data can therefore be stored untagged, for a smaller serialisation size, and be written and read by an Avro library.

  - At the time of writing, Avro libraries are available for C, C++, C#, Java, Python, Ruby and PHP. Optionally tools are available for C++, Java and C# to generate code for encoding and decoding.

- Google Protocol Buffers [15]. Data is stored according to a pre-defined schema. The schema is used by a generator to generate code for encoding and decoding the data. Data can therefore be stored untagged, for a smaller serialisation size. The schema is not stored with the data, so unlike Avro cannot be read with a generic library.

  - Code must be generated for a particular data schema to to read and write data using that schema. At the time of writing, the Google code generator can currently generate code for encoding and decoding a schema for C++, Go, Java, Python, Ruby, C#, Objective-C, Javascript and PHP.

- CBOR [16]. Defined in [RFC7049], this serialisation format is comparable to JSON but with a binary representation. It does not use a pre-defined schema, so data is always stored tagged. However, CBOR data schemas can be described using CDDL [I-D.greevenbosch-appsawg-cbor-cddl] and tools exist to verify data files conform to the schema.

  - CBOR is a simple format, and simple to implement. At the time of writing, the CBOR website lists implementations for 16 languages.

Avro and Protocol Buffers both allow storage of untagged data, but because they rely on the data schema for this, their implementation is considerably more complex than CBOR. Using Avro or Protocol Buffers in an unsupported environment would require notably greater development effort compared to CBOR.

A test program was written which reads input from a PCAP file and writes output using one of two basic structures; either a simple structure, where each query/response pair is represented in a single record entry, or the C-DNS block structure.

The resulting output files were then compressed using a variety of common general-purpose lossless compression tools to explore the compressibility of the formats. The compression tools employed were:

- snzip [17]. A command line compression tool based on the Google Snappy [18] library.

- lz4 [19]. The command line compression tool from the reference C LZ4 implementation.

- gzip [20]. The ubiquitous GNU zip tool.

- zstd [21]. Compression using the Zstandard algorithm.

- xz [22]. A popular compression tool noted for high compression.

In all cases the compression tools were run using their default settings.

Note that this draft does not mandate the use of compression, nor any particular compression scheme, but it anticipates that in practice output data will be subject to general-purpose compression, and so this should be taken into consideration.

"test.pcap", a 662Mb capture of sample data from a root instance was used for the comparison. The following table shows the formatted size and size after compression (abbreviated to Comp. in the table headers), together with the task resident set size (RSS) and the user time taken by the compression. File sizes are in Mb, RSS in kb and user time in seconds.

Format	File size	Comp.	Comp. size	RSS	User time
PCAP	661.87	snzip	212.48	2696	1.26
		lz4	181.58	6336	1.35
		gzip	153.46	1428	18.20
		zstd	87.07	3544	4.27
		xz	49.09	97416	160.79
JSON simple	4113.92	snzip	603.78	2656	5.72
		lz4	386.42	5636	5.25
		gzip	271.11	1492	73.00
		zstd	133.43	3284	8.68
		xz	51.98	97412	600.74
Avro simple	640.45	snzip	148.98	2656	0.90
		lz4	111.92	5828	0.99
		gzip	103.07	1540	11.52
		zstd	49.08	3524	2.50
		xz	22.87	97308	90.34

CBOR simple	764.82	snzip	164.57	2664	1.11
		lz4	120.98	5892	1.13
		gzip	110.61	1428	12.88
		zstd	54.14	3224	2.77
		xz	23.43	97276	111.48
PBuf simple	749.51	snzip	167.16	2660	1.08
		lz4	123.09	5824	1.14
		gzip	112.05	1424	12.75
		zstd	53.39	3388	2.76
		xz	23.99	97348	106.47
JSON block	519.77	snzip	106.12	2812	0.93
		lz4	104.34	6080	0.97
		gzip	57.97	1604	12.70
		zstd	61.51	3396	3.45
		xz	27.67	97524	169.10
Avro block	60.45	snzip	48.38	2688	0.20
		lz4	48.78	8540	0.22
		gzip	39.62	1576	2.92
		zstd	29.63	3612	1.25
		xz	18.28	97564	25.81
CBOR block	75.25	snzip	53.27	2684	0.24
		lz4	51.88	8008	0.28
		gzip	41.17	1548	4.36
		zstd	30.61	3476	1.48
		xz	18.15	97556	38.78
PBuf block	67.98	snzip	51.10	2636	0.24
		lz4	52.39	8304	0.24
		gzip	40.19	1520	3.63
		zstd	31.61	3576	1.40
		xz	17.94	97440	33.99

The above results are discussed in the following sections.

C.1. Comparison with full PCAP files

An important first consideration is whether moving away from PCAP offers significant benefits.

The simple binary formats are typically larger than PCAP, even though they omit some information such as Ethernet MAC addresses. But not only do they require less CPU to compress than PCAP, the resulting compressed files are smaller than compressed PCAP.

C.2. Simple versus block coding

The intention of the block coding is to perform data de-duplication on query/response records within the block. The simple and block formats above store exactly the same information for each query/response record. This information is parsed from the DNS traffic in the input PCAP file, and in all cases each field has an identifier and the field data is typed.

The data de-duplication on the block formats show an order of magnitude reduction in the size of the format file size against the simple formats. As would be expected, the compression tools are able to find and exploit a lot of this duplication, but as the de-duplication process uses knowledge of DNS traffic, it is able to retain a size advantage. This advantage reduces as stronger compression is applied, as again would be expected, but even with the strongest compression applied the block formatted data remains around 75% of the size of the simple format and its compression requires roughly a third of the CPU time.

C.3. Binary versus text formats

Text data formats offer many advantages over binary formats, particularly in the areas of ad-hoc data inspection and extraction. It was therefore felt worthwhile to carry out a direct comparison, implementing JSON versions of the simple and block formats.

Concentrating on JSON block format, the format files produced are a significant fraction of an order of magnitude larger than binary formats. The impact on file size after compression is as might be expected from that starting point; the stronger compression produces files that are 150% of the size of similarly compressed binary format, and require over 4x more CPU to compress.

C.4. Performance

Concentrating again on the block formats, all three produce format files that are close to an order of magnitude smaller that the original "test.pcap" file. CBOR produces the largest files and Avro the smallest, 20% smaller than CBOR.

However, once compression is taken into account, the size difference narrows. At medium compression (with gzip), the size difference is 4%. Using strong compression (with xz) the difference reduces to 2%, with Avro the largest and Protocol Buffers the smallest, although CBOR and Protocol Buffers require slightly more compression CPU.

The measurements presented above do not include data on the CPU required to generate the format files. Measurements indicate that writing Avro requires 10% more CPU than CBOR or Protocol Buffers. It appears, therefore, that Avro's advantage in compression CPU usage is probably offset by a larger CPU requirement in writing Avro.

C.5. Conclusions

The above assessments lead us to the choice of a binary format file using blocking.

As noted previously, this draft anticipates that output data will be subject to compression. There is no compelling case for one particular binary serialisation format in terms of either final file size or machine resources consumed, so the choice must be largely based on other factors. CBOR was therefore chosen as the binary serialisation format for the reasons listed in Section 6.

C.6. Block size choice

Given the choice of a CBOR format using blocking, the question arises of what an appropriate default value for the maximum number of query/response pairs in a block should be. This has two components; what is the impact on performance of using different block sizes in the format file, and what is the impact on the size of the format file before and after compression.

The following table addresses the performance question, showing the impact on the performance of a C++ program converting "test.pcap" to C-DNS. File size is in Mb, resident set size (RSS) in kb.

Block size	File size	RSS	User time
1000	133.46	612.27	15.25
5000	89.85	676.82	14.99
10000	76.87	752.40	14.53
20000	67.86	750.75	14.49
40000	61.88	736.30	14.29
80000	58.08	694.16	14.28
160000	55.94	733.84	14.44
320000	54.41	799.20	13.97

Increasing block size, therefore, tends to increase maximum RSS a little, with no significant effect (if anything a small reduction) on CPU consumption.

The following figure plots the effect of increasing block size on output file size for different compressions.

Figure showing effect of block size on file size (PNG) [23]

Figure showing effect of block size on file size (SVG) [24]

From the above, there is obviously scope for tuning the default block size to the compression being employed, traffic characteristics, frequency of output file rollover etc. Using a strong compression, block sizes over 10,000 query/response pairs would seem to offer limited improvements.

Authors' Addresses

John Dickinson
Sinodun IT
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA

Email: jad@sinodun.com

Jim Hague
Sinodun IT
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA

Email: jim@sinodun.com

Sara Dickinson
Sinodun IT
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA

Email: sara@sinodun.com

Terry Manderson
ICANN
12025 Waterfront Drive
Suite 300
Los Angeles CA 90094-2536

Email: terry.manderson@icann.org

John Bond
ICANN
12025 Waterfront Drive
Suite 300
Los Angeles CA 90094-2536

Email: john.bond@icann.org