dnsop
Internet-Draft
Intended status: Standards Track
Expires: June 9, 2017

J. Dickinson
J. Hague
S. Dickinson
Sinodun IT
T. Manderson
J. Bond
ICANN

December 6, 2016

C-DNS: A DNS Packet Capture Format
draft-ietf-dnsop-dns-capture-format-00

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 meta data. 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 June 9, 2017.

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 DITL [ditl].

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

There has also been work on using other text based formats to describe DNS packets [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 100's 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.

This documents 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 to minimise the size

This document contains

- A discussion of the some common use cases in which such DNS data is collected. See Section 3.
- A discussion of the major design considerations in developing an efficient data representation for collections of DNS messages. See Section 4.
- A definition of a CBOR [RFC7049] representation of a collection of DNS messages. This will be referred to as the C-DNS format (Compacted-DNS). See Section 7.
- Notes on converting C-DNS back to PCAP format. See Section 8.
- Some high level implementation considerations for applications designed to produce C-DNS, e.g. a query response matching algorithm. See Section 9.

2. Requirements 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].

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.

- Servers are hosted in a variety of situations
  - Operator self hosted servers
  - Third party hosting (including multiple third parties)
  - Third party hardware (including multiple third parties)
- 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 attack

    * On servers that are unwitting intermediaries in attacks

- Traffic can be collected via a variety of mechanisms

    * On the same hardware as the name server itself

    * Using a network tap to listen in from another server

    * Using port mirroring to listen in from another server

- 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 on the in-band network

Clearly, 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 unfeasible to reliably collect full PCAPS especially if the server is under attack. In the latter case, collection of full PCAPs may be reasonable.

As a result of these restrictions the data format discussed below 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 are multiple third party servers involved.

and therefore minimal storage size of the capture files is a major factor.

Another 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 is the most important job of a name server. Any data collection system co-located with the name server will need to be intelligent enough to carefully manage its CPU, disk, memory and network utilization. Hence this use case benefits from a format that has 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 resulting collection format should facilitate the re-creation of common formats (such as PCAPs) 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.

- 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 Query 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 lowers storage requirements due to commonality in the data in most cases.

- 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 PCAPS. 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.

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

Rationale: This blocking and indexing provides a significant reduction in the volume of file data generated. Whilst introducing complexity it provides compression of the data that makes use of knowledge of the DNS packet structure.

[TODO: Further discussion on commonality between DNS packets e.g.

- common query signatures

- for the authoritative case there are a finite set of valid responses and much commonality in NXDOMAIN responses]

It is anticipated that the files produced will 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 D.

- Meta-data about other packets received should also 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 are of interest.

It should be noted that any structured capture format that does not capture the DNS payload byte for byte will likely be limited to some extent in that it cannot represent 'malformed' DNS packets. Only those packets that can be transformed reasonably into the structured format can be represented by it. So if a query is malformed this will lead to the (well formed) DNS responses with error code FORMERR appearing as 'unmatched'.

[TODO: Need further discussion of well-formed vs malformed packets and how name servers view this definition.]

[TODO: Need to develop optional representation of malformed packets within CBOR and what this means for packet matching. This may influence which fields are optional in the rest of the representation.]

Packets such as those described above can be separately recorded in a PCAP file for later analysis.

5. C-DNS 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 so economical in storage space.
- Other similar representations were investigated, and whilst all had attractive features, none had a significant advantage over CBOR. See Appendix C and Appendix D - for some discussion of this.
- CBOR is an IETF Standard and familiar to IETF participants, and being based on the successful JSON text format, requires very little familiarization for those in the wider industry.
- CBOR can also be easily converted to JSON for debugging and other human inspection requirements.
- CBOR data schemas can be described using CDDL [I-D.greevenbosch-appsawg-cbor-cddl].

7. C-DNS CBOR 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 preamble. The preamble contains information on the collection settings.

This 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 pairs detailing the queries and responses received during the block. The list of Q/R pairs 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 in the low 1000's give good results. See Appendix D.2 for more details.

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

In the following

- For all quantities that contain bit flags, bit 0 indicates the least significant bit.

- Items described as indexes are the index of the data item in the referenced table. Indexes are 1-based. An index value of 0 is reserved to mean not present.

7.3. File header contents

The file header contains the following:

Field	Type	Description
File type ID	Text string	String 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
Format version	Unsigned	Indicates version of format used in file.
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. Blank if converting an existing PCAP 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.
Snap length	Unsigned	Collect up to this many bytes per packet.
Promiscuous mode	Unsigned	1 if promiscuous mode was enabled on the interface, 0 otherwise.
Interfaces	Array of text strings	Identifiers of the interfaces used for collection.

VLAN IDs	Array of unsigned	Identifiers of VLANs selected for collection.
Filter	Text string	"tcpdump" [pcap] style filter for input.
Query collection options	Unsigned	Bit flags indicating sections in Query packets to be collected. Bit 0. Collect second and subsequent question sections. Bit 1. Collect Answer sections. Bit 2. Collect Authority sections. Bit 3. Collection Additional sections.
Response collection options	Unsigned	Bit flags indicating sections in Response packets to be collected. Bit 0. Collect second and subsequent question sections. 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.

7.6. Block contents

Each block contains the following:

Field	Type	Description
Block preamble	Map of items	Overall information for the block.
Block statistics	Map of statistics	Statistics about the block.
Block tables	Map of tables	The tables containing data referenced by individual Q/R entries.
Q/Rs	Array of Q/Rs	Details of individual Q/R pairs.
Address Event Counts	Array of Address Event counts	Per client counts of ICMP messages and TCP resets.

7.7. Block preamble map

The block preamble map contains overall information for the block.

Field	Type	Description
Timestamp	Array of unsigned	A timestamp for the earliest record in the block. The timestamp is specified as a CBOR array with two elements 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 latter is always a value between 0 and 999,999.

[TODO: Extend to support pico/nano. Also do this for Time offset and Response delay]

7.8. Block statistics

[TODO: Add block 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 pair 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 pair contains a Query. If the pair contains a Response only, the field will not be present.

7.10. IP address table

This table holds all client and server IP addresses in the block. Each item in the table is a single IP address.

Field	Type	Description
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

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

Field	Description
Class	CLASS value.
Type	TYPE value.

[TODO: Can this be optimized? Should a class of IN be inferred if not present?]

7.12. Name/RDATA table

This table 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.

Field	Type	Description
Data	Byte string	The NAME or RDATA contents. NAMEs, and labels within RDATA contents, are in uncompressed label format.

7.13. Query Signature table

This table holds elements of the Q/R data that are often common to between different individual Q/R records. Each item in the table is a CBOR map. Each item in the map has an unsigned value and an unsigned 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	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 protocol 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.
Q/R signature flags	A	Bit flags indicating information present in this Q/R pair. 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.
Q/R 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) flag. Bit 1. Query Authenticated Data (AD) flag. Bit 2. Query reserved (Z) flag. Bit 3. Query Recursion Available (RA) flag. Bit 4. Query Recursion Desired (RD) flag. Bit 5. Query TrunCation (TC) flag. Bit 6. Query Authoritative Answer (AA) flag. Bit 7. Query DNSSEC answer OK (D0) flag. Bit 8. Response Checking Disabled (CD) flag. Bit 9. Response Authenticated Data (AD) flag. Bit 10. Response reserved (Z) flag. Bit 11. Response Recursion Available (RA) flag. Bit 12. Response Recursion Desired (RD) flag. Bit 13. Response TrunCation (TC) flag. Bit 14. Response Authoritative Answer (AA)

		flag.
Query RCODE	Q	Query RCODE. If the Query contains OPT, this value incorporates any EXTENDED_RCODE_VALUE.
Question Class/Type	QT	The index in the Class/Type table of the CLASS and TYPE of the first Question.
Question QDCOUNT	QT	The QDCOUNT in the Query, or Response if no Query present.
Query ANCOUNT	Q	Query ANCOUNT.
Query ARCOUNT	Q	Query ARCOUNT.
Query NSCOUNT	Q	Query NSCOUNT.
Query EDNS version	QO	The Query EDNS version.
EDNS UDP size	QO	The Query EDNS sender's UDO payload size
Query OPT RDATA	QO	The index in the NAME/RDATA table of the OPT RDATA.
Response RCODE	R	Response RCODE. If the Response contains OPT, this value incorporates any EXTENDED_RCODE_VALUE.

7.14. Question table

This table 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 key. This data is optionally collected.

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

7.15. Resource Record (RR) table

This table 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	The index in the NAME/RDATA table of the NAME.
Class/Type	The index in the Class/Type table of the CLASS and TYPE of the RR.
TTL	The RR Time to Live.
RDATA	The index in the NAME/RDATA table of the RR RDATA.

7.16. Question list table

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

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

7.17. Resource Record list table

This table holds a list of individual Resource Records in a Answer, Authority or Additional section. Each item in the table is a CBOR unsigned. 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 items. Each item in the array is a CBOR map containing details on the individual Q/R pair.

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 key.

- Query extended information and Response extended information which are of Type Extended Information.
- Response delay which is an integer (This can be negative if the network stack/capture library returns them out of order.)

Field	P	Description
Time offset	A	Q/R timestamp as an offset in microseconds from the Block pre-amble Timestamp. The timestamp is the timestamp of the Query, or the Response if there is no Query.
Client address	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	A	The index of the more information on the Q/R in the Query Signature table.
Client hoplimit	Q	The IPv4 TTL or IPv6 Hoplimit from the Query packet.
Response delay	QR	The time different between Query and Response, in microseconds.
Question NAME	QT	The index in the NAME/RDATA table of the QNAME for the first Question.
Response size	R	The size of the DNS message (not the packet containing the message, just the DNS message) that forms the Response.
Query extended information	Q	Extended Query information. This item is only present if collection of extra Query information is configured.
Response extended information	R	Extended Response information. This item is only present if collection of extra Query information is configured.

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

Field	Description
Question	The index in the Questions list table of the entry listing the second and subsequent Question sections for the Query or Response.
Answer	The index in the RR list table of the entry listing the Answer Resource Record sections for the Query or Response.
Authority	The index in the RR list table of the entry listing the Authority Resource Record sections for the Query or Response.
Additional	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
Event 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.
Event code	Unsigned	A code relating to the event. Optional.
Address index	Unsigned	The index in the IP address table of the client address.
Count	Unsigned	The number of occurrences of this event during the block collection period.

8. C-DNS to PCAP

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

Firstly 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 is imperfect.

Secondly, even if all sections of the response were captured name compression presents a challenge in reconstructing the DNS response payload byte for byte. Section 8.1 discusses this is more detail.

Thirdly, not all transport information is 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 and non-DNS packets

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 the malformed and non-DNS packets are captured separately into PCAPs they can be merged with PCAPs reconstructed from C-DNS to produce a more complete packet stream.

8.1. 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 re-produce 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 packet. 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.

From the C-DNS format it is not possible to ensure that the DNS response payload is reconstructed byte for byte. 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-BOR this is the best that can be achieved.

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

9. 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 etc. has already been performed

2. Each message is associated with transport meta-data required to generate the Primary ID (see below)

3. Each message has a well-formed DNS header of 12 bytes and (if present) the first RR in the query 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.

o 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.]

9.1. Matching algorithm

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

Figure showing the packet matching algorithm format (PNG) [5]

Figure showing the packet matching algorithm format (SVG) [6]

and further details of the algorithm are given in the following sections.

9.2. Message identifiers

9.2.1. Primary ID (required)

A Primary ID can be constructed for each message which 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

9.2.2. 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, RCODE=FORMERR or NOTIMP)

9.3. Algorithm Parameters

1. Configurable timeout

9.4. 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.

9.5. Algorithm Limitations

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

9.6. Workspace

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

9.7. 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 items have one of three types of content:

1. Paired Q/R messages

2. A query message (no response)

3. A response message (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.

9.8. Post Processing

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

10. IANA Considerations

None

11. Security Considerations

Any control interface MUST perform authentication and encryption.

Any data upload MUST be authenticated and encrypted.

12. 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.

Thanks to Robert Edmonds, Paul Hoffman and Jerry Lundstroem for review.

Also, Miek Gieben for mmark [7]

13. Changelog

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 CBOR 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

14. References

14.1. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7049>.

14.2. 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] Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-09 (work in progress), September 2016.

[I-D.hoffman-dns-in-json] Hoffman, P., "Representing DNS Messages in JSON", draft-hoffman-dns-in-json-09 (work in progress), October 2016.

[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>.

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

14.3. URIs

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

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

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

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

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

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

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

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

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

Appendix A. CDDL

```
; CDDL specification of the file format for C-DNS,
; which describes a collection of DNS Query/Response pairs.

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

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

format-version = 0
configuration  = 1
generator-id   = 2
host-id        = 3
```

```
Configuration = {
    ? query-timeout    => uint,
    ? skew-timeout     => uint,
    ? snaplen          => uint,
    ? promisc          => uint,
    ? interfaces       => [* tstr],
    ? vlan-ids         => [* uint],
    ? filter           => tstr,
    ? query-options    => uint,    ; See below
    ? response-options => uint,
    ? accept-rr-types  => [* tstr],
    ? ignore-rr-types  => [* tstr],
}

; query-options and response-options are bitmasks. A bit set adds in the
; specified sections.
;
; second & subsequent question sections = 1
; answer sections = 2
; authority sections = 4
; additional sections = 8

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;


Block = {
    preamble             => BlockPreamble,
    ? statistics         => BlockStatistics, ; Much of this could be derived
    tables               => BlockTables,
    queries              => [* QueryResponse],
    address-event-counts => [* AddressEventCount],
}

preamble             = 0
statistics           = 1
tables               = 2
queries              = 3
address-event-counts = 4
```

```
BlockPreamble = {
    start-time => Timeval
}

start-time = 1

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

BlockStatistics = {
    ? total-packets        => uint,
    ? total-pairs          => uint,
    ? unmatched_queries    => uint,
    ? unmatched_responses  => uint,
    ? malformed-packets    => uint,
    ? non-dns-packets      => uint,
    ? out-of-order-packets => uint,
    ? missing-pairs        => uint,
    ? missing-packets      => uint,
    ? missing-non-dns      => uint,
}

total-packets        = 0
total-pairs          = 1
unmatched_queries    = 2
unmatched_responses  = 3
malformed-packets    = 4
non-dns-packets      = 5
out-of-order-packets = 6
missing-pairs        = 7
missing-packets      = 8
missing-non-dns      = 9

BlockTables = {
    ip-address => [* bstr],
    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 earliest record
    client-address-index  => uint,
    client-port           => uint,
    transaction-id        => uint,
    query-signature-index => uint,
    ? client-hoplimit     => uint,
    ? delay-useconds      => int,        ; Times may be -ve at capture
    ? query-name-index    => uint,
    ? response-size       => uint,       ; DNS size of response
    ? query-extended      => QueryResponseExtended,
    ? response-extended   => QueryResponseExtended,
}

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

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

type  = 0
class = 1

QuerySignature = {
    server-address-index    => uint,
    server-port             => uint,
    transport-flags         => uint,
    qr-sig-flags            => uint,
    ? query-opcode          => uint,
    qr-dns-flags            => uint,
```

```
    ? 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,
)
```

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 - 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 [8] 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 [9] 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:

   a 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.

   b The compression target name MUST be in RDATA.

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

   d 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 representations were considered in particular CBOR, Apache Avro and Protocol Buffers.

Protocol Buffers and Avro both require a data schema, and validate data being stored against that schema.

[TODO: Finish pros and cons of CBOR vs Avro vs Protocol buffers - tools, schema, adoption, etc.]

The difference in file sizes were mostly minimal See Appendix D.3.

Appendix D. Sample data on the C-DNS format

This section presents some example figures for the output size of capture files when using different block sizes, data representations and binary formats. The data is sample data for a root instance.

[TODO: This section needs more work..]

D.1. Comparison to full PCAPS

As can be seen in more detail below for this sample data the compressed C-DNS files are around 30% the size of the full compressed PCAPs. It should also be noted that experiments showed that compression of the C-DNS format required very roughly an order of magnitude less CPU resources than compression of full PCAPSs when using one core from a 3.5GHz i7 processor.

D.2. Block size choices

[TODO: Discuss trade-off of file block size vs memory consumption.]

[TODO: Add graph that demonstrates block size of 5000 is optimal for the sample data used.]

D.3. Blocking vs more simple output

Some experiments were conducted producing output in a very simple format involving a single record per Q/R data item (akin to a .csv representation). The aim here was to examine whether the blocking mechanism (using a block size of 5000) was worth the complexity, particularly after compression of the output file using several general purpose compression tools. The original PCAP file was 325.79M and compressed using xz to 24.3Mb.

Format	Output size	lz4	gzip	xz
cbor-simple	44.23M	16.06M	11.50M	7.51M
cbor-block	22.44M	15.14M	10.70M	7.23M

It might be expected that blocking is exploiting commonality that a general purpose compression engine could also exploit, and the figures do indeed bear this out. The more powerful (and resource-consuming) the compression, the closer the compressed simple file size gets to the compressed chunk file size. With no compression, the blocked output size is typically half that of the simple output, but as greater degrees of compression are applied the gap shrinks. However, even with the stronger compressor, the chunked output remains roughly 5-10% smaller than the simple output. This, and the higher gains at lower compression, might be significant, depending on the target environment.

[TODO: Add data on reduction in CPU overhead of compressing blocked output vs simple output.]

This was repeated using some other binary representations:

Format	Output size	lz4	gzip	xz
json-simple	189.85M	25.59M	16.03M	9.74M
avro-simple	43.31M	16.07M	11.92M	7.99M
avro-block	17.44M	12.94M	10.08M	7.18M
protobuf-simple	46.02M	15.79M	11.59M	7.94M
protobuf-block	22.08M	15.43M	10.91M	7.40M

There's not a lot to choose between the three contenders with simple output. Avro produces the smaller output, CBOR the next and Protocol Buffers the largest, but the different is under 10%. However, with blocking, while CBOR and Protocol Buffers are again within a few percentage points of each other (though Protocol Buffers now has a slight advantage), Avro produces files in the region of 20% smaller, and holds a diminishing advantage through increased compression.

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