Internet Engineering Task Force (IETF)
Request for Comments: 8618
Category: Standards Track
ISSN: 2070-1721

J. Dickinson
J. Hague
S. Dickinson
Sinodun IT
T. Manderson
ICANN
J. Bond
Wikimedia Foundation, Inc.

September 2019

Compacted-DNS (C-DNS): A Format for DNS Packet Capture

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 is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8618.

Table of Contents

1. Introduction 4
2. Terminology 5
3. Data Collection Use Cases 5
4. Design Considerations 8
5. Choice of CBOR 10
6. C-DNS Format Conceptual Overview 10
6.1. Block Parameters 14
6.2. Storage Parameters 14
6.2.1. Optional Data Items 15
6.2.2. Optional RRs and OPCODEs 16
6.2.3. Storage Flags 17
6.2.4. IP Address Storage 17
7. C-DNS Format Detailed Description 18
7.1. Map Quantities and Indexes 18
7.2. Tabular Representation 18
7.3. "File" 19
7.3.1. "FilePreamble" 20
7.3.1.1. "BlockParameters" 20
7.3.1.1.1. "StorageParameters" 21
7.3.1.1.1.1. "StorageHints" 22
7.3.1.1.2. "CollectionParameters" 24
7.3.2. "Block" 25
7.3.2.1. "BlockPreamble" 26
7.3.2.2. "BlockStatistics" 27
7.3.2.3. "BlockTables" 28
7.3.2.3.1. "ClassType" 29
7.3.2.3.2. "QueryResponseSignature" 30
7.3.2.3.3. "Question" 33
7.3.2.3.4. "RR" 34
7.3.2.3.5. "MalformedMessageData" 34

7.3.2.4. "QueryResponse" 35
7.3.2.4.1. "ResponseProcessingData" 36
7.3.2.4.2. "QueryResponseExtended" 37
7.3.2.5. "AddressEventCount" 38
7.3.2.6. "MalformedMessage" 39
8. Versioning 39
9. C-DNS to PCAP 40
9.1. Name Compression 42
10. Data Collection 42
10.1. Matching Algorithm 43
10.2. Message Identifiers 45
10.2.1. Primary ID (Required) 45
10.2.2. Secondary ID (Optional) 46
10.3. Algorithm Parameters 46
10.4. Algorithm Requirements 46
10.5. Algorithm Limitations 47
10.6. Workspace 47
10.7. Output 47
10.8. Post-Processing 47
11. Implementation Guidance 47
11.1. Optional Data 48
11.2. Trailing Bytes 48
11.3. Limiting Collection of RDATA 49
11.4. Timestamps 49
12. IANA Considerations 49
12.1. Transport Types 49
12.2. Data Storage Flags 50
12.3. Response-Processing Flags 51
12.4. AddressEvent Types 51
13. Security Considerations 52
14. Privacy Considerations 52
15. References 53
15.1. Normative References 53
15.2. Informative References 55
Appendix A. CDDL 58
Appendix B. DNS Name Compression Example 69
B.1. NSD Compression Algorithm 70
B.2. Knot Authoritative Compression Algorithm 70
B.3. Observed Differences 71
Appendix C. Comparison of Binary Formats 71
C.1. Comparison with Full PCAP Files 74
C.2. Simple versus Block Coding 74
C.3. Binary versus Text Formats 75
C.4. Performance 75
C.5. Conclusions 75
C.6. Block Size Choice 76

Appendix D. Data Fields for Traffic Regeneration 77
D.1. Recommended Fields for Traffic Regeneration 77
D.2. Issues with Small Data Captures 77
Acknowledgements 78
Authors' Addresses 79

1. Introduction

There has long been a need for server operators 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 the DNS Statistics Collector (DSC) [dsc], packetq [packetq], dnscap [dnscap], and dnstap [dnstap]. However, there is no standard exchange format for large DNS packet captures. The PCAP ("packet capture") [pcap] format or the PCAP Next Generation (PCAP-NG) [pcapng] format is 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. Additionally, these tools and formats typically have no filter mechanism to selectively record only certain fields at capture time, requiring post-processing for anonymization or pseudonymization of data to protect user privacy.

There has also been work on using text-based formats to describe DNS packets (for example, see [dnsxml] and [RFC8427]), but this work is 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 and transmission size (and therefore upload overhead) of the data collected is highly desirable.

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

- o Minimize the file size for storage and transmission.

- o Minimize 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 some common use cases in which 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 description of why the Concise Binary Object Representation (CBOR) [RFC7049] was chosen for this format; see Section 5.

- A conceptual overview of the C-DNS format; see Section 6.

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

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

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

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

"Packet" refers to an individual IPv4 or IPv6 packet. Typically, packets are UDP datagrams, but such packets may also be part of a TCP data stream. "Message", unless otherwise qualified, refers to a DNS payload extracted from a UDP datagram or a TCP data stream.

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

3. Data Collection Use Cases

From a purely server operator perspective, collecting full packet captures of all packets going into or out of a name server provides the most comprehensive picture of network activity. 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:

  * Within the name server implementation itself

  * 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 out-of-band networks with the same capacity as the in-band network). In the former case, 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 is designed with the most limited use case in mind, such that:

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

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

- o Collected data being returned to some central analysis system will use the same network interface as the DNS Queries and Responses

- o 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 is also essential that interoperability with less restricted infrastructure is 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 "Query/Response (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 the associated Response represent the basic level of a client's 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.

   In the context of generating a C-DNS file, it is assumed that only those DNS payloads that can be parsed to produce a well-formed DNS message are stored in the structured Query/Response data items of the C-DNS format and that all other messages will (optionally) be recorded as separate malformed messages. Parsing a well-formed message means, at a minimum, the following:

   * The packet has a well-formed 12-byte DNS Header with a recognized OPCODE.

   * The section counts are consistent with the section contents.

   * All of the Resource Records (RRs) can be fully parsed.

2. All top-level fields in each Query/Response data item will be optional.

   * Rationale: Different operators will have different requirements for data to be available for analysis. Operators with minimal requirements should not have to pay the cost of recording full data, though this will limit the ability to perform certain kinds of data analysis and also to reconstruct packet captures. For example, omitting the RRs from a Response will reduce the C-DNS file size; in principle, Responses can be synthesized if there is enough context. Operators may have different policies for collecting user data and can choose to omit or anonymize certain fields at capture time, e.g., client address.

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

   * Rationale: This blocking and indexing action 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.

   * Examples of commonality between DNS messages are that in most cases the QUESTION RR is the same in the Query and Response and that there is a finite set of Query "signatures" (based on a subset of attributes). For many authoritative servers, there is very likely to be a finite set of Responses that are generated, of which a large number are NXDOMAIN.

4. Traffic metadata can optionally be included in each block. Specifically, counts of some types of non-DNS packets (e.g., ICMP, TCP resets) sent to the server may be of interest.

5. The wire-format content of malformed DNS messages may 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 messages. Only those messages that can be fully parsed and transformed into the structured format can be fully represented. Note, however, that this can result in rather misleading statistics. For example, a malformed Query that cannot be represented in the C-DNS format will lead to the (well-formed) DNS Response with error code FORMERR appearing as "unmatched". Therefore, it can greatly aid downstream analysis to have the wire format of the malformed DNS messages available directly in the C-DNS file.

5. Choice of CBOR

This document presents a detailed format description for C-DNS. The format uses CBOR [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 specification and is 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. Formats that are more complex require library support, which may present problems on unusual platforms.

- CBOR can also be easily converted to text formats such as JSON [RFC8259] for debugging and other human inspection requirements.

- CBOR data schemas can be described using the Concise Data Definition Language (CDDL) [RFC8610].

6. C-DNS Format 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 1 shows the C-DNS format at the top level, including the file header and data blocks. The Query/Response data items, Address/Event Count data items, and Malformed Message data items link to various Block Tables.

```
                   +-------+
                   + C-DNS |
                   +-------+--------------------------+
                   | File Type Identifier             |
                   +----------------------------------+
                   | File Preamble                    |
                   | +--------------------------------+
                   | | Format Version                 |
                   | +--------------------------------+
                   | | Block Parameters               |
                   +-+--------------------------------+
                   | Block                            |
                   | +--------------------------------+
                   | | Block Preamble                 |
                   | +--------------------------------+
                   | | Block Statistics               |
                   | +--------------------------------+
                   | | Block Tables                   |
                   | +--------------------------------+
                   | | Query/Response data items      |
                   | +--------------------------------+
                   | | Address/Event Count data items |
                   | +--------------------------------+
                   | | Malformed Message data items   |
                   +-+--------------------------------+
                   | Block                            |
                   | +--------------------------------+
                   | | Block Preamble                 |
                   | +--------------------------------+
                   | | Block Statistics               |
                   | +--------------------------------+
                   | | Block Tables                   |
                   | +--------------------------------+
                   | | Query/Response data items      |
                   | +--------------------------------+
                   | | Address/Event Count data items |
                   | +--------------------------------+
                   | | Malformed Message data items   |
                   +-+--------------------------------+
                   | Further Blocks...                |
                   +----------------------------------+
```

Figure 1: The C-DNS Format

Figure 2 shows some more-detailed relationships within each Block, specifically those between the Query/Response data item and the relevant Block Tables. Some fields have been omitted for clarity.

```
   +----------------+
   | Query/Response |
   +-------------------------+
   | Time Offset             |
   +-------------------------+            +------------------+
   | Client Address          |---------+->| IP Address array |
   +-------------------------+         |  +------------------+
   | Client Port             |         |
   +-------------------------+         |  +------------------+
   | Transaction ID          |     +---)->| Name/RDATA array |<--------+
   +-------------------------+     |   |  +------------------+         |
   | Query Signature         |--+  |   |                               |
   +-------------------------+  |  |   |  +-----------------+          |
   | Client Hoplimit (q)     |  +--)---)->| Query Signature |          |
   +-------------------------+     |   |  +-----------------+--------+  |
   | Response Delay (r)      |     |   +--| Server Address          |  |
   +-------------------------+     |      +-------------------------+  |
   | Query Name              |--+--+      | Server Port             |  |
   +-------------------------+  |         +-------------------------+  |
   | Query Size (q)          |  |         | Transport Flags         |  |
   +-------------------------+  |         +-------------------------+  |
   | Response Size (r)       |  |         | QR Type                 |  |
   +-------------------------+  |         +-------------------------+  |
   | Response Processing (r) |  |         | QR Signature Flags      |  |
   | +-----------------------+  |         +-------------------------+  |
   | | Bailiwick             |--+         | Query OPCODE (q)        |  |
   | +-----------------------+            +-------------------------+  |
   | | Flags                 |            | QR DNS Flags            |  |
   +-+-----------------------+            +-------------------------+  |
   | Extra Query Info (q)    |            | Query RCODE (q)         |  |
   | +-----------------------+            +-------------------------+  |
   | | Question              |--+---+  +--+-Query Class/Type (q)    |  |
   | +-----------------------+      |  |  +-------------------------+  |
   | | Answer                |--+   |  |  | Query QDCOUNT (q)       |  |
   | +-----------------------+  |   |  |  +-------------------------+  |
   | | Authority             |--+   |  |  | Query ANCOUNT (q)       |  |
   | +-----------------------+  |   |  |  +-------------------------+  |
   | | Additional            |--+   |  |  | Query NSCOUNT (q)       |  |
```

```
   +-+-----------------------+  |   |  |  +-------------------------+  |
   | Extra Response Info (r) |  |-+ |  |  | Query ARCOUNT (q)       |  |
   | +-----------------------+  | | |  |  +-------------------------+  |
   | | Answer                |--+ | |  |  | Query EDNS version (q)  |  |
   | +-----------------------+  | | |  |  +-------------------------+  |
   | | Authority             |--+ | |  |  | Query EDNS UDP Size (q) |  |
   | +-----------------------+  | | |  |  +-------------------------+  |
   | | Additional            |--+ | |  |  | Query OPT RDATA (q)     |--+
   +-+-----------------------+    | |  |  +-------------------------+  |
                                  | |  |  | Response RCODE (r)      |  |
                                  | |  |  +-------------------------+  |
   + -----------------------------+ |  +----------+                    |
   |                                |             |                    |
   | + -----------------------------+             |                    |
   | |  +---------------+  +----------+           |                    |
   | +->| Question List |->| Question |           |                    |
   |    | array         |  | array    |           |                    |
   |    +---------------+  +----------+--+        |                    |
   |                       | Name        |--+-----)--------------------+
   |                       +-------------+  |     |  +------------+
   |                       | Class/Type  |--)---+-+->| Class/Type |
   |                       +-------------+  |   |    | array      |
   |                                        |   |    +------------+--+
   |                                        |   |    | CLASS         |
   |    +---------------+  +----------+     |   |    +---------------+
   +--->| RR List array |->| RR array |     |   |    | TYPE          |
        +---------+-----+  +----------+--+  |   |    +---------------+
                           | Name        |--+   |
                           +-------------+      |
                           | Class/Type  |------+
                           +-------------+
```

Figure 2: The Query/Response Data Item and Subsidiary Tables

In Figure 2, data items annotated (q) are only present when a Query/Response has a Query, and those annotated (r) are only present when a Query/Response Response is present.

A C-DNS file begins with a file header containing a File Type Identifier and a File Preamble. The File Preamble contains information on the file Format Version and an array of Block Parameters items (the contents of which include Collection and Storage Parameters used for one or more Blocks).

The file header is followed by a series of Blocks.

A Block consists of a Block Preamble item, some Block Statistics for the traffic stored within the Block, and then various arrays of common data collectively called the Block Tables. This is then followed by an array of the Query/Response data items detailing the Queries and Responses stored within the Block. The array of Query/Response data items is in turn followed by the Address/Event Count data items (an array of per-client counts of particular IP events) and then Malformed Message data items (an array of malformed messages that are stored in the Block).

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 Query/Response data items give good results. See Appendix C.6 for more details.

This design exploits data commonality and block-based storage to minimize the C-DNS file size. As a result, C-DNS cannot be streamed below the level of a Block.

6.1. Block Parameters

The details of the Block Parameters items are not shown in the diagrams but are discussed here for context.

An array of Block Parameters items is stored in the File Preamble (with a minimum of one item at index 0); a Block Parameters item consists of a collection of Storage and Collection Parameters that applies to any given Block. An array is used in order to support use cases such as wanting to merge C-DNS files from different sources. The Block Preamble item then contains an optional index for the Block Parameters item that applies for that Block; if not present, the index defaults to 0. Hence, in effect, a global Block Parameters item is defined that can then be overridden per Block.

6.2. Storage Parameters

The Block Parameters item includes a Storage Parameters item -- this contains information about the specific data fields stored in the C-DNS file.

These parameters include:

- o The sub-second timing resolution used by the data.

- o Information (hints) on which optional data are omitted. See Section 6.2.1.

- Recorded OPCODES [opcodes] and RR TYPEs [rrtypes]. See Section 6.2.2.

- Flags indicating, for example, whether the data is sampled or anonymized. See Sections 6.2.3 and 14.

- Client and server IPv4 and IPv6 address prefixes. See Section 6.2.4.

6.2.1. Optional Data Items

To enable implementations to store data to their precise requirements in as space-efficient a manner as possible, all fields in the following arrays are optional:

- Query/Response

- Query Signature

- Malformed Messages

In other words, an implementation can choose to omit any data item that is not required for its use case (whilst observing the restrictions relating to IP address storage described in Section 6.2.4). In addition, implementations may be configured to not record all RRs or to only record messages with certain OPCODES.

This does, however, mean that a consumer of a C-DNS file faces two problems:

1. How can it quickly determine if a file definitely does not contain the data items it requires to complete a particular task (e.g., reconstructing DNS traffic or performing a specific piece of data analysis)?

2. How can it determine whether a data item is not present because it was (1) explicitly not recorded or (2) not available/present?

For example, capturing C-DNS data from within a name server implementation makes it unlikely that the Client Hoplimit can be recorded. Or, if there is no Query ARCOUNT recorded and no Query OPT RDATA [RFC6891] recorded, is that because no Query contained an OPT RR, or because that data was not stored?

The Storage Parameters item therefore also contains a Storage Hints item, which specifies which items the encoder of the file omits from the stored data and will therefore never be present. (This approach is taken because a flag that indicated which items were included for collection would not guarantee that the item was present -- only that it might be.) An implementation decoding that file can then use these flags to quickly determine whether the input data is not rich enough for its needs.

One scenario where this may be particularly important is the case of regenerating traffic. It is possible to collect such a small set of data items that an implementation decoding the file cannot determine if a given Query/Response data item was generated from just a Query, just a Response, or a Query/Response pair. This makes it impossible to reconstruct DNS traffic even if sensible defaults are provided for the missing data items. This is discussed in more detail in Section 9.

6.2.2. Optional RRs and OPCODEs

Also included in the Storage Parameters item are explicit arrays listing the RR TYPEs and the OPCODEs to be recorded. These arrays remove any ambiguity over whether, for example, messages containing particular OPCODEs are not present because (1) certain OPCODEs did not occur or (2) the implementation is not configured to record them.

In the case of OPCODEs, for a message to be fully parsable, the OPCODE must be known to the collecting implementation. Any message with an OPCODE unknown to the collecting implementation cannot be validated as correctly formed and so must be treated as malformed. Messages with OPCODES known to the recording application but not listed in the Storage Parameters item are discarded by the recording application during C-DNS capture (regardless of whether they are malformed or not).

In the case of RRs, each record in a message must be fully parsable, including parsing the record RDATA, as otherwise the message cannot be validated as correctly formed. Any RR with an RR TYPE not known to the collecting implementation cannot be validated as correctly formed and so must be treated as malformed.

Once a message is correctly parsed, an implementation is free to record only a subset of the RRs present.

6.2.3. Storage Flags

The Storage Parameters item contains flags that can be used to indicate if:

- the data is anonymized,
- the data is produced from sample data, or
- names in the data have been normalized (converted to uniform case).

The Storage Parameters item also contains optional fields holding details of the sampling method used and the anonymization method used. It is RECOMMENDED that these fields contain URIs [RFC3986] pointing to resources describing the methods used. See Section 14 for further discussion of anonymization and normalization.

6.2.4. IP Address Storage

The format can store either full IP addresses or just IP prefixes; the Storage Parameters item contains fields to indicate if only IP prefixes were stored.

If the IP address prefixes are absent, then full addresses are stored. In this case, the IP version can be directly inferred from the stored address length and the fields "qr-transport-flags" in QueryResponseSignature, "ae-transport-flags" in AddressEventCount, and "mm-transport-flags" in MalformedMessageData (which contain the IP version bit) are optional.

If IP address prefixes are given, only the prefix bits of addresses are stored. In this case, in order to determine the IP version, the fields "qr-transport-flags" in QueryResponseSignature, "ae-transport-flags" in AddressEventCount, and "mm-transport-flags" in MalformedMessageData MUST be present. See Sections 7.3.2.3.2 and 7.3.2.3.5.

As an example of storing only IP prefixes, if a client IPv6 prefix of 48 is specified, a client address of 2001:db8:85a3::8a2e:370:7334 will be stored as 0x20010db885a3, reducing address storage space requirements. Similarly, if a client IPv4 prefix of 16 is specified, a client address of 192.0.2.1 will be stored as 0xc000 (192.0).

7. C-DNS Format Detailed Description

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

7.1. Map Quantities and Indexes

All map keys are integers with values specified in the CDDL. String keys would significantly bloat the file size.

All key values specified are positive integers under 24, so their CBOR representation is a single byte. Positive integer values not currently used as keys in a map are reserved for use in future standard extensions.

Implementations may choose to add additional implementation-specific entries to any map. Negative integer map keys are reserved for these values. Key values from -1 to -24 also have a single-byte CBOR representation, so such implementation-specific extensions are not at any space efficiency disadvantage.

An item described as an index is the index of the data item in the referenced array. Indexes are 0-based.

7.2. Tabular Representation

The following sections present the C-DNS specification in tabular format with a detailed description of each item.

In all quantities that contain bit flags, bit 0 indicates the least significant bit, i.e., flag "n" in quantity "q" is on if "(q & (1 << n)) != 0".

For the sake of readability, all type and field names defined in the CDDL definition are shown in double quotes. Type names are by convention camel case (e.g., "BlockTables"), and field names are lowercase with hyphens (e.g., "block-tables").

For the sake of brevity, the following conventions are used in the tables:

- The column M marks whether items in a map are mandatory.
  * X - Mandatory items.
  * C - Conditionally mandatory items. Such items are usually optional but may be mandatory in some configurations.
  * If the column is empty, the item is optional.

- The column T gives the CBOR datatype of the item.
  * U - Unsigned integer.
  * I - Signed integer (i.e., either a CBOR unsigned integer or a CBOR negative integer).
  * B - Boolean.
  * S - Byte string.
  * T - Text string.
  * M - Map.
  * A - Array.

In the case of maps and arrays, more information on the type of each value, including the CDDL definition name if applicable, is given in the description.

7.3. "File"

A C-DNS file has an outer structure "File", an array that contains the following:

Field	M	T	Description
file-type-id	X	T	String "C-DNS" identifying the file type.
file-preamble	X	M	Version and parameter information for the whole file. Map of type "FilePreamble"; see Section 7.3.1.
file-blocks	X	A	Array of items of type "Block"; see Section 7.3.2. The array may be empty if the file contains no data.

7.3.1. "FilePreamble"

Information about data in the file. A map containing the following:

Field	M	T	Description
major-format-version	X	U	Unsigned integer "1". The major version of the format used in the file. See Section 8.
minor-format-version	X	U	Unsigned integer "0". The minor version of the format used in the file. See Section 8.
private-version		U	Version indicator available for private use by implementations.
block-parameters	X	A	Array of items of type "BlockParameters". See Section 7.3.1.1. The array must contain at least one entry. (The "block-parameters-index" item in each "BlockPreamble" indicates which array entry applies to that "Block".)

7.3.1.1. "BlockParameters"

Parameters relating to data storage and collection that apply to one or more items of type "Block". A map containing the following:

Field	M	T	Description
storage-parameters	X	M	Parameters relating to data storage in a "Block" item. Map of type "StorageParameters"; see Section 7.3.1.1.1.
collection-parameters		M	Parameters relating to collection of the data in a "Block" item. Map of type "CollectionParameters"; see Section 7.3.1.1.2.

7.3.1.1.1. "StorageParameters"

Parameters relating to how data is stored in the items of type "Block". A map containing the following:

Field	M	T	Description
ticks-per-second	X	U	Sub-second timing is recorded in ticks. This specifies the number of ticks in a second.
max-block-items	X	U	The maximum number of items stored in any of the arrays in a "Block" item (Q/R, Address/Event Count, or Malformed Message data items). An indication to a decoder of the resources needed to process the file.
storage-hints	X	M	Collection of hints as to which fields are omitted in the arrays that have optional fields. Map of type "StorageHints". See Section 7.3.1.1.1.1.
opcodes	X	A	Array of OPCODES [opcodes] (unsigned integers, each in the range 0 to 15 inclusive) recorded by the collecting implementation. See Section 6.2.2.
rr-types	X	A	Array of RR TYPEs [rrtypes] (unsigned integers, each in the range 0 to 65535 inclusive) recorded by the collecting implementation. See Section 6.2.2.
storage-flags		U	Bit flags indicating attributes of stored data. Bit 0. 1 if the data has been anonymized. Bit 1. 1 if the data is sampled data. Bit 2. 1 if the names have been normalized (converted to uniform case).
client-address -prefix-ipv4		U	IPv4 client address prefix length, in the range 1 to 32 inclusive. If specified, only the address prefix bits are stored.

Field	M	T	Description
client-address -prefix-ipv6		U	IPv6 client address prefix length, in the range 1 to 128 inclusive. If specified, only the address prefix bits are stored.
server-address -prefix-ipv4		U	IPv4 server address prefix length, in the range 1 to 32 inclusive. If specified, only the address prefix bits are stored.
server-address -prefix-ipv6		U	IPv6 server address prefix length, in the range 1 to 128 inclusive. If specified, only the address prefix bits are stored.
sampling-method		T	Information on the sampling method used. See Section 6.2.3.
anonymization -method		T	Information on the anonymization method used. See Section 6.2.3.

7.3.1.1.1.1. "StorageHints"

An indicator of which fields the collecting implementation omits in the maps with optional fields. Note that hints have a top-down precedence. In other words, where a map contains another map, the hint on the containing map overrides any hints in the contained map and the contained map is omitted. A map containing the following:

Field	M	T	Description
query-response -hints	X	U	Hints indicating which "QueryResponse" fields are omitted; see Section 7.3.2.4. If a bit is unset, the field is omitted from the capture. Bit 0. time-offset Bit 1. client-address-index Bit 2. client-port Bit 3. transaction-id Bit 4. qr-signature-index Bit 5. client-hoplimit Bit 6. response-delay Bit 7. query-name-index Bit 8. query-size Bit 9. response-size

			Bit 10. response-processing-data Bit 11. query-question-sections Bit 12. query-answer-sections Bit 13. query-authority-sections Bit 14. query-additional-sections Bit 15. response-answer-sections Bit 16. response-authority-sections Bit 17. response-additional-sections
query-response-signature-hints	X	U	Hints indicating which "QueryResponseSignature" fields are omitted; see Section 7.3.2.3.2. If a bit is unset, the field is omitted from the capture. Bit 0. server-address-index Bit 1. server-port Bit 2. qr-transport-flags Bit 3. qr-type Bit 4. qr-sig-flags Bit 5. query-opcode Bit 6. qr-dns-flags Bit 7. query-rcode Bit 8. query-classtype-index Bit 9. query-qdcount Bit 10. query-ancount Bit 11. query-nscount Bit 12. query-arcount Bit 13. query-edns-version Bit 14. query-udp-size Bit 15. query-opt-rdata-index Bit 16. response-rcode
rr-hints	X	U	Hints indicating which optional "RR" fields are omitted; see Section 7.3.2.3.4. If a bit is unset, the field is omitted from the capture. Bit 0. ttl Bit 1. rdata-index
other-data-hints	X	U	Hints indicating which other datatypes are omitted. If a bit is unset, the datatype is omitted from the capture. Bit 0. malformed-messages Bit 1. address-event-counts

7.3.1.1.2. "CollectionParameters"

Parameters providing information regarding how data in the file was collected (applicable for some, but not all, collection environments). The values are informational only and serve as metadata to downstream analyzers as to the configuration of a collecting implementation. They can provide context when interpreting what data is present/absent from the capture but cannot necessarily be validated against the data captured.

These parameters have no default. If they do not appear, nothing can be inferred about their value.

A map containing the following items:

Field	M	T	Description
query-timeout		U	To be matched with a Query, a Response must arrive within this number of milliseconds.
skew-timeout		U	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 microseconds.
snaplen		U	Collect up to this many bytes per packet.
promisc		B	"true" if promiscuous mode [pcap-options] was enabled on the interface, "false" otherwise.
interfaces		A	Array of identifiers (of type text string) of the interfaces used for collection.
server-addresses		A	Array of server collection IP addresses (of type byte string). Metadata for downstream analyzers; does not affect collection.

vlan-ids		A	Array of identifiers (of type unsigned integer, each in the range 1 to 4094 inclusive) of VLANs [IEEE802.1Q] selected for collection. VLAN IDs are unique only within an administrative domain.
filter		T	Filter for input, in "tcpdump" [pcap-filter] style.
generator-id		T	Implementation-specific human-readable string identifying the collection method.
host-id		T	String identifying the collecting host.

7.3.2. "Block"

Container for data with common collection and storage parameters. A map containing the following:

Field	M	T	Description
block-preamble	X	M	Overall information for the "Block" item. Map of type "BlockPreamble"; see Section 7.3.2.1.
block-statistics		M	Statistics about the "Block" item. Map of type "BlockStatistics"; see Section 7.3.2.2.
block-tables		M	The arrays containing data referenced by individual "QueryResponse" or "MalformedMessage" items. Map of type "BlockTables"; see Section 7.3.2.3.
query-responses		A	Details of individual C-DNS Q/R data items. Array of items of type "QueryResponse"; see Section 7.3.2.4. If present, the array must not be empty.

address-event -counts		A	Per-client counts of ICMP messages and TCP resets. Array of items of type "AddressEventCount"; see Section 7.3.2.5. If present, the array must not be empty.
malformed-messages		A	Details of malformed DNS messages. Array of items of type "MalformedMessage"; see Section 7.3.2.6. If present, the array must not be empty.

7.3.2.1. "BlockPreamble"

Overall information for a "Block" item. A map containing the following:

Field	M	T	Description
earliest-time	C	A	A timestamp (two unsigned integers, of type "Timestamp") for the earliest record in the "Block" item. The first integer is the number of seconds since the POSIX epoch [posix-time] ("time_t"), excluding leap seconds. The second integer is the number of ticks (see Section 7.3.1.1.1) since the start of the second. This field is mandatory unless all block items containing a time offset from the start of the Block also omit that time offset.
block-parameters -index		U	The index of the item in the "block-parameters" array (in the "file-preamble" item) applicable to this block. If not present, index 0 is used. See Section 7.3.1.

7.3.2.2. "BlockStatistics"

Basic statistical information about a "Block" item. A map containing the following:

Field	M	T	Description
processed-messages		U	Total number of well-formed DNS messages processed from the input traffic stream during collection of data in this "Block" item.
qr-data-items		U	Total number of Q/R data items in this "Block" item.
unmatched-queries		U	Number of unmatched Queries in this "Block" item.
unmatched-responses		U	Number of unmatched Responses in this "Block" item.
discarded-opcode		U	Number of DNS messages processed from the input traffic stream during collection of data in this "Block" item but not recorded because their OPCODE is not in the list to be collected.
malformed-items		U	Number of malformed messages processed from the input traffic stream during collection of data in this "Block" item.

7.3.2.3. "BlockTables"

Map of arrays containing data referenced by individual "QueryResponse" or "MalformedMessage" items in this "Block". Each element is an array that, if present, must not be empty.

An item in the "qlist" array contains indexes to values in the "qrr" array. Therefore, if "qlist" is present, "qrr" must also be present. Similarly, if "rrlist" is present, "rr" must also be present.

The map contains the following items:

Field	M	T	Description
ip-address		A	Array of IP addresses, in network byte order (of type byte string). If client or server address prefixes are set, only the address prefix bits are stored. Each string is therefore up to 4 bytes long for an IPv4 address, or up to 16 bytes long for an IPv6 address. See Section 7.3.1.1.1.
classtype		A	Array of RR CLASS and TYPE information. Type is "ClassType". See Section 7.3.2.3.1.
name-rdata		A	Array where each entry is the contents of a single NAME or RDATA in wire format (of type byte string). Note that NAMEs, and labels within RDATA contents, are full domain names or labels; no name compression (per [RFC1035]) is used on the individual names/labels within the format.
qr-sig		A	Array of Q/R data item signatures. Type is "QueryResponseSignature". See Section 7.3.2.3.2.
qlist		A	Array of type "QuestionList". A "QuestionList" is an array of unsigned integers, indexes to "Question" items in the "qrr" array.

qrr		A	Array of type "Question". Each entry is the contents of a single Question, where a Question is the second or subsequent Question in a Query. See Section 7.3.2.3.3.
rrlist		A	Array of type "RRList". An "RRList" is an array of unsigned integers, indexes to "RR" items in the "rr" array.
rr		A	Array of type "RR". Each entry is the contents of a single RR. See Section 7.3.2.3.4.
malformed-message -data		A	Array of the contents of malformed messages. Array of type "MalformedMessageData". See Section 7.3.2.3.5.

7.3.2.3.1. "ClassType"

RR CLASS and TYPE information. A map containing the following:

Field	M	T	Description
type	X	U	TYPE value [rrtypes].
class	X	U	CLASS value [rrclasses].

7.3.2.3.2. "QueryResponseSignature"

Elements of a Q/R data item that are often common between multiple individual Q/R data items. A map containing the following:

Field	M	T	Description
server-address-index		U	The index in the "ip-address" array of the server IP address. See Section 7.3.2.3.
server-port		U	The server port.
qr-transport-flags	C	U	Bit flags describing the transport used to service the Query. Same definition as "mm-transport-flags" in Section 7.3.2.3.5, with an additional indicator for trailing bytes. See Appendix A. Bit 0. IP version. 0 if IPv4, 1 if IPv6. See Section 6.2.4. Bits 1-4. Transport. 4-bit unsigned value where 0 = UDP [RFC1035] 1 = TCP [RFC1035] 2 = TLS [RFC7858] 3 = DTLS [RFC8094] 4 = HTTPS [RFC8484] 15 = Non-standard transport (see below) Values 5-14 are reserved for future use. Bit 5. 1 if trailing bytes in Query packet. See Section 11.2.
qr-type		U	Type of Query/Response transaction based on the definitions in the dnstap schema [dnstap-schema]. 0 = Stub. A transaction between a stub resolver and a DNS server from the perspective of the stub resolver. 1 = Client. A transaction between a client and a DNS server (a proxy or full recursive resolver) from the perspective of the DNS server.

			2 = Resolver. A transaction between a recursive resolver and an authoritative server from the perspective of the recursive resolver. 3 = Authoritative. A transaction between a recursive resolver and an authoritative server from the perspective of the authoritative server. 4 = Forwarder. A transaction between a downstream forwarder and an upstream DNS server (a recursive resolver) from the perspective of the downstream forwarder. 5 = Tool. A transaction between a DNS software tool and a DNS server, from the perspective of the tool.
qr-sig-flags		U	Bit flags explicitly indicating attributes of the message pair represented by this Q/R data item (not all attributes may be recorded or deducible). Bit 0. 1 if a Query was present. Bit 1. 1 if a Response was present. Bit 2. 1 if a Query was present and it had an OPT RR. Bit 3. 1 if a Response was present and it had an OPT RR. Bit 4. 1 if a Query was present but had no Question. Bit 5. 1 if a Response was present but had no Question (only one query-name-index is stored per Q/R data item).
query-opcode		U	Query OPCODE.
qr-dns-flags		U	Bit flags with values from the Query and Response DNS flags. 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		U	Query RCODE. If the Query contains an OPT RR [RFC6891], this value incorporates any EXTENDED-RCODE value [rcodes].
query-classtype-index		U	The index in the "classtype" array of the CLASS and TYPE of the first Question. See Section 7.3.2.3.
query-qdcount		U	The QDCOUNT in the Query, or Response if no Query present.
query-ancount		U	Query ANCOUNT.
query-nscount		U	Query NSCOUNT.
query-arcount		U	Query ARCOUNT.
query-edns-version		U	The Query EDNS version. ("EDNS" stands for Extension Mechanisms for DNS.)
query-udp-size		U	The Query EDNS sender's UDP payload size.

query-opt-rdata-index		U	The index in the "name-rdata" array of the OPT RDATA. See Section 7.3.2.3.
response-rcode		U	Response RCODE. If the Response contains an OPT RR [RFC6891], this value incorporates any EXTENDED-RCODE value [rcodes].

Version 1.0 of C-DNS supports transport values corresponding to DNS transports defined in IETF Standards Track documents at the time of writing. There are numerous non-standard methods of sending DNS messages over various transports using a variety of protocols, but they are out of scope for this document. With the current specification, these can be generically stored using value 15 (Non-standard transport), or implementations are free to use the negative integer map keys to define their own mappings. Such non-standard transports may also be the subject of a future extension to the specification.

7.3.2.3.3. "Question"

Details on individual Questions in a Question section. A map containing the following:

Field	M	T	Description
name-index	X	U	The index in the "name-rdata" array of the QNAME. See Section 7.3.2.3.
classtype-index	X	U	The index in the "classtype" array of the CLASS and TYPE of the Question. See Section 7.3.2.3.

7.3.2.3.4. "RR"

Details on individual RRs in RR sections. A map containing the following:

Field	M	T	Description
name-index	X	U	The index in the "name-rdata" array of the NAME. See Section 7.3.2.3.
classtype-index	X	U	The index in the "classtype" array of the CLASS and TYPE of the RR. See Section 7.3.2.3.
ttl		U	The RR Time to Live.
rdata-index		U	The index in the "name-rdata" array of the RR RDATA. See Section 7.3.2.3.

7.3.2.3.5. "MalformedMessageData"

Details on malformed DNS messages stored in this "Block" item. A map containing the following:

Field	M	T	Description
server-address -index		U	The index in the "ip-address" array of the server IP address. See Section 7.3.2.3.
server-port		U	The server port.
mm-transport-flags	C	U	Bit flags describing the transport used to service the Query. See Section 6.2.4. Bits 1-4. Transport. 4-bit unsigned value where 0 = UDP [RFC1035] 1 = TCP [RFC1035] 2 = TLS [RFC7858] 3 = DTLS [RFC8094] 4 = HTTPS [RFC8484] 15 = Non-standard transport Values 5-14 are reserved for future use.

mm-payload		S	The payload (raw bytes) of the DNS message.

7.3.2.4. "QueryResponse"

Details on individual Q/R data items.

Note that there is no requirement that the elements of the "query-responses" array are presented in strict chronological order.

A map containing the following items:

Field	M	T	Description
time-offset		U	Q/R timestamp as an offset in ticks (see Section 7.3.1.1.1) from "earliest-time". The timestamp is the timestamp of the Query, or the Response if there is no Query.
client-address-index		U	The index in the "ip-address" array of the client IP address. See Section 7.3.2.3.
client-port		U	The client port.
transaction-id		U	DNS transaction identifier.
qr-signature-index		U	The index in the "qr-sig" array of the "QueryResponseSignature" item. See Section 7.3.2.3.
client-hoplimit		U	The IPv4 TTL or IPv6 Hoplimit from the Query packet.
response-delay		I	The time difference between Query and Response, in ticks. See Section 7.3.1.1.1. Only present if there is a Query and a Response. The delay can be negative if the network stack/capture library returns packets out of order.

query-name-index		U	The index in the "name-rdata" array of the item containing the QNAME for the first Question. See Section 7.3.2.3.
query-size		U	DNS Query message size (see below).
response-size		U	DNS Response message size (see below).
response-processing-data		M	Data on Response processing. Map of type "ResponseProcessingData". See Section 7.3.2.4.1.
query-extended		M	Extended Query data. Map of type "QueryResponseExtended". See Section 7.3.2.4.2.
response-extended		M	Extended Response data. Map of type "QueryResponseExtended". See Section 7.3.2.4.2.

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. For TCP, it is the size of the DNS message as specified in the two-byte message length header. Trailing bytes in UDP Queries are routinely observed in traffic to authoritative servers, and this value allows a calculation of how many trailing bytes were present.

7.3.2.4.1. "ResponseProcessingData"

Information on the server processing that produced the Response. A map containing the following:

Field	M	T	Description
bailiwick-index		U	The index in the "name-rdata" array of the owner name for the Response bailiwick. See Section 7.3.2.3.
processing-flags		U	Flags relating to Response processing. Bit 0. 1 if the Response came from cache.

7.3.2.4.2. "QueryResponseExtended"

Extended data on the Q/R data item.

Each item in the map is present only if collection of the relevant details is configured.

A map containing the following items:

Field	M	T	Description
question-index		U	The index in the "qlist" array of the entry listing any second and subsequent Questions in the Question section for the Query or Response. See Section 7.3.2.3.
answer-index		U	The index in the "rrlist" array of the entry listing the Answer RR sections for the Query or Response. See Section 7.3.2.3.
authority-index		U	The index in the "rrlist" array of the entry listing the Authority RR sections for the Query or Response. See Section 7.3.2.3.
additional-index		U	The index in the "rrlist" array of the entry listing the Additional RR sections for the Query or Response. See Section 7.3.2.3. Note that Query OPT RR data can optionally be stored in the QuerySignature.

7.3.2.5. "AddressEventCount"

Counts of various IP-related events relating to traffic with individual client addresses. A map containing the following:

Field	M	T	Description
ae-type	X	U	The type of event. The following event 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		U	A code relating to the event. For ICMP or ICMPv6 events, this MUST be the ICMP [RFC792] or ICMPv6 [RFC4443] code. For other events, the contents are undefined.
ae-transport-flags	C	U	Bit flags describing the transport used to service the event. See Section 6.2.4. Bit 0. IP version. 0 if IPv4, 1 if IPv6. Bits 1-4. Transport. 4-bit unsigned value where 0 = UDP [RFC1035] 1 = TCP [RFC1035] 2 = TLS [RFC7858] 3 = DTLS [RFC8094] 4 = HTTPS [RFC8484] 15 = Non-standard transport Values 5-14 are reserved for future use.
ae-address-index	X	U	The index in the "ip-address" array of the client address. See Section 7.3.2.3.
ae-count	X	U	The number of occurrences of this event during the Block collection period.

7.3.2.6. "MalformedMessage"

Details on Malformed Message data items. A map containing the following:

Field	M	T	Description
time-offset		U	Message timestamp as an offset in ticks (see Section 7.3.1.1.1) from "earliest-time".
client-address-index		U	The index in the "ip-address" array of the client IP address. See Section 7.3.2.3.
client-port		U	The client port.
message-data-index		U	The index in the "malformed-message-data" array of the message data for this message. See Section 7.3.2.3.

8. Versioning

The C-DNS File Preamble includes a file Format Version; a major and minor version number are required fields. This document defines version 1.0 of the C-DNS specification. This section describes the intended use of these version numbers in future specifications.

It is noted that version 1.0 includes many optional fields; therefore, consumers of version 1.0 should be inherently robust to parsing files with variable data content.

Within a major version, a new minor version MUST be a strict superset of the previous minor version, with no semantic changes to existing fields. New keys MAY be added to existing maps, and new maps MAY be added. A consumer capable of reading a particular major.minor version MUST also be capable of reading all previous minor versions of the same major version. It SHOULD also be capable of parsing all subsequent minor versions, ignoring any keys or maps that it does not recognize.

A new major version indicates changes to the format that are not backwards compatible with previous major versions. A consumer capable of only reading a particular major version (greater than 1) is neither required nor expected to be capable of reading a previous major version.

9. C-DNS to PCAP

It is usually possible to reconstruct 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 of well-formed DNS messages depends on two factors:

1. Whether or not a particular subset of the optional fields were captured in the C-DNS file, specifically the data fields necessary to reconstruct a valid IP header and DNS payload for both Query and Response (see Appendix D.1). Clearly, if not all these data fields were captured, the reconstruction is likely to be imperfect even if reasonable defaults are provided for the reconstruction.

2. Whether or not at least one field was captured that unambiguously identifies the Query/Response data item as containing just a Query, just a Response, or a Query/Response pair. Obviously, the qr-sig-flags defined in Section 7.3.2.3.2 is such a field; however, this field is optional. For more details, see Appendix D.2.

It is noted again that simply having hints that indicate that certain data fields were not omitted does not guarantee that those data fields were actually captured. Therefore, the ability to reconstruct PCAP data (in the absence of defaults) can in principle vary for each record captured in a C-DNS file, and between Blocks that have differing hints.

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 in 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 reconstructed 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 been split across multiple TCP segments

  * Multiple DNS messages may have been sent on a single TCP session

- TLS session information:

  * TLS version or cipher suites

  * TLS-related features such as TCP Fast Open (TFO) [RFC7413] or TLS session resumption [RFC5077]

- DNS-over-HTTPS [RFC8484] message details:

  * Whether the message used POST or GET

  * HTTPS Headers

- 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 messages 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 name compression (per [RFC1035]) 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.

Name compression per [RFC1035] 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 Query from different name server software that 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 that the correct algorithm has been used, as it is possible to match the length whilst still not matching the on-the-wire bytes; however, 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 production of a stream of Q/R data items, matching Queries and Responses where possible.

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

1. All IP fragmentation reassembly, TCP stream reassembly, and so on, have 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 preprocessing stage, but the correctly formed Response with RCODE of FORMERR being present.

DNS messages are processed in the order they are delivered to the implementation.

It should be noted that packet capture libraries do not necessarily provide packets in strict chronological order. This can, for example, arise on multi-core platforms where packets arriving at a network device are processed by different cores. On systems where this behavior has been observed, the timestamps associated with each packet are consistent; Queries always have a timestamp prior to the Response timestamp. However, the order in which these packets appear in the packet capture stream is not necessarily strictly chronological; a Response can appear in the capture stream before the Query that provoked the Response. For this discussion, this non-chronological delivery is termed "skew".

In the presence of skew, Response packets can arrive for matching before the corresponding Query. To avoid generating false instances of Responses without a matching Query, and Queries without a matching Response, the matching algorithm must take the possibility of skew into account.

10.1. Matching Algorithm

A schematic representation of the algorithm for matching Q/R data items is shown in Figure 3. It takes individual DNS Query or Response messages as input, and it outputs matched Q/R data items. The numbers in the figure identify matching operations listed in Table 1. Specific details of the algorithm -- for example, queues, timers, and identifiers -- are given in the following sections.

```
                       .----------------------.
                       | Process next message |<------------------+
                       `----------------------'                   |
                                   |                              |
                   +------------------------------+               |
                   | Generate message identifiers |               |
                   +------------------------------+               |
                                   |                              |
                          Response | Query                        |
                   +--------------< >---------------+             |
                   |                                |             |
         +--------------------+           +--------------------+  |
         | Find earliest QR   |           | Create QR item (2) |  |
         | item in OFIFO (1)  |           +--------------------+  |
         +--------------------+                     |             |
                    |                       +---------------+     |
              Match | No match              | Append new QR |     |
          +--------< >------+               | item to OFIFO |     |
          |                 |               +---------------+     |
    +-----------+      +--------+                   |             |
    | Update QR |      | Add to |          +-------------------+  |
    | item (3)  |      | RFIFO  |          | Find earliest QR  |  |
    +-----------+      +--------+          | item in RFIFO (1) |  |
          |                 |              +-------------------+  |
          +-----------------+                       |             |
                    |                               |             |
                    |     +----------------+  Match | No match    |
                    |     | Remove R       |-------< >-----+      |
                    |     | from RFIFO (3) |               |      |
                    |     +----------------+               |      |
                    |              |                       |      |
                    +--------------+-----------------------+      |
                                   |                              |
            +----------------------------------------------+      |
            | Update all timed-out (QT) OFIFO QR items (4) |      |
            +----------------------------------------------+      |
                                   |                              |
                   +--------------------------------+             |
                   | Remove all timed-out (ST) R    |             |
                   | from RFIFO, create QR item (5) |             |
                   +--------------------------------+             |
               ____________________|_______________________       |
              /                                            /      |
             /  Remove all consecutive done entries from  /-------+
            /   front of OFIFO for further processing    /
           /____________________________________________/
```

OFIFO = output FIFO containing Q/R data items (Section 10.6)
RFIFO = Response FIFO containing unmatched Response items (Section 10.6)
QT = Query Timeout (Section 10.3)
ST = Skew Timeout (Section 10.3)

Figure 3: Query/Response Matching Algorithm

Reference	Operation
(1)	Find earliest QR item in FIFO where: * QR.done = false * QR.Q.PrimaryID == R.PrimaryID and, if both QR.Q and R have SecondaryID: * QR.Q.SecondaryID == R.SecondaryID
(2)	Set: QR.Q := Q QR.R := nil QR.done := false
(3)	Set: QR.R := R QR.done := true
(4)	Set: QR.done := true
(5)	Set: QR.Q := nil QR.R := R QR.done := true

Table 1: Operations Used in the Matching Algorithm

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 (QT). A Query arrives with timestamp t1. If no Response matching that Query has arrived before other input arrives timestamped later than (t1 + QT), a Q/R data item containing only a Query is recorded. The QT value is typically on the order of 5 seconds.

2. Skew Timeout (ST). A Response arrives with timestamp t2. If a Response has not been matched by a Query before input arrives timestamped later than (t2 + ST), a Q/R data item containing only a Response is recorded. The ST value is typically a few microseconds.

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

The algorithm employs two FIFO queues:

- o OFIFO: an output FIFO containing Q/R data items in chronological order.

- o RFIFO: a FIFO holding Responses without a matching Query in order of arrival.

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 a capture, all items in the RFIFO are timed out immediately, generating Response only entries to the OFIFO. These and all other remaining entries in the OFIFO should be treated as timed-out Queries.

11. Implementation Guidance

Whilst this document makes no specific recommendations with respect to "Canonical CBOR" (see Section 3.9 of [RFC7049]), the following guidance may be of use to implementers.

Adherence to the first two rules given in Section 3.9 of [RFC7049] will minimize file sizes.

Adherence to the last two rules given in Section 3.9 of [RFC7049] for all maps and arrays would unacceptably constrain implementations -- for example, in the use case of real-time data collection in constrained environments where outputting Block Tables after Q/R data items and allowing indefinite-length maps and arrays could reduce memory requirements.

It is recommended that implementations that have fundamental restrictions on what data fields they can collect SHOULD always store hints with the bits unset for those fields, i.e., they unambiguously indicate that those data fields will be omitted from captured C-DNS.

11.1. Optional Data

When decoding C-DNS data, some of the items required for a particular function that the consumer wishes to perform may be missing. Consumers should consider providing configurable default values to be used in place of the missing values in their output.

11.2. Trailing Bytes

A DNS Query message in a UDP or TCP payload can be followed by some additional (spurious) bytes, which are not stored in C-DNS.

When DNS traffic is sent over TCP, each message is prefixed with a two-byte length field, which gives the message length, excluding the two-byte length field. In this context, trailing bytes can occur in two circumstances, with different results:

1. The number of bytes consumed by fully parsing the message is less than the number of bytes given in the length field (i.e., the length field is incorrect and too large). In this case, the surplus bytes are considered trailing bytes in a manner analogous to UDP and recorded as such. If only this case occurs, it is possible to process a packet containing multiple DNS messages where one or more have trailing bytes.

2. There are surplus bytes between the end of a well-formed message and the start of the length field for the next message. In this case, the first of the surplus bytes will be processed as the first byte of the next length field, and parsing will proceed from there, almost certainly leading to the next and any subsequent messages in the packet being considered malformed. This will not generate a trailing-bytes record for the processed well-formed message.

11.3. Limiting Collection of RDATA

Implementations should consider providing a configurable maximum RDATA size for captures -- for example, to avoid memory issues when confronted with large zone transfer records.

11.4. Timestamps

The preamble to each block includes a timestamp of the earliest record in the Block. As described in Section 7.3.2.1, the timestamp is an array of two unsigned integers. The first is a POSIX "time_t" [posix-time]. Consumers of C-DNS should be aware of this, as it excludes leap seconds and therefore may cause minor anomalies in the data, e.g., when calculating Query throughput.

12. IANA Considerations

IANA has created a registry "C-DNS DNS Capture Format" containing the subregistries defined in Sections 12.1 to 12.4 inclusive.

In all cases, new entries may be added to the subregistries by Expert Review as defined in [RFC8126]. Experts are expected to exercise their own expert judgment and should consider the following general guidelines in addition to any provided guidelines that are particular to a subregistry.

- o There should be a real and compelling use for any new value.
- o Values assigned should be carefully chosen to minimize storage requirements for common cases.

12.1. Transport Types

IANA has created a registry "C-DNS Transports" of C-DNS transport type identifiers. The primary purpose of this registry is to provide unique identifiers for all transports used for DNS Queries.

The following note is included in this registry: "In version 1.0 of C-DNS [RFC8618], there is a field to identify the type of DNS transport. This field is 4 bits in size."

The initial contents of the registry are as follows. See Sections 7.3.2.3.2, 7.3.2.3.5, and 7.3.2.5 of this document:

Identifier	Name	Reference
0	UDP	RFC 8618
1	TCP	RFC 8618
2	TLS	RFC 8618
3	DTLS	RFC 8618
4	HTTPS	RFC 8618
5-14	Unassigned	
15	Non-standard transport	RFC 8618

Expert reviewers should take the following point into consideration: Is the requested DNS transport described by a Standards Track RFC?

12.2. Data Storage Flags

IANA has created a registry "C-DNS Storage Flags" of C-DNS data storage flags. The primary purpose of this registry is to provide indicators giving hints on processing of the data stored.

The following note is included in this registry: "In version 1.0 of C-DNS [RFC8618], there is a field describing attributes of the data recorded. The field is a CBOR [RFC7049] unsigned integer holding bit flags."

The initial contents of the registry are as follows. See Section 7.3.1.1.1 of this document:

Bit	Name	Description	Reference
0	anonymized-data	The data has been anonymized.	RFC 8618
1	sampled-data	The data is sampled data.	RFC 8618
2	normalized-names	Names in the data have been normalized.	RFC 8618
3-63	Unassigned		

12.3. Response-Processing Flags

IANA has created a registry "C-DNS Response Flags" of C-DNS response-processing flags. The primary purpose of this registry is to provide indicators giving hints on the generation of a particular Response.

The following note is included in this registry: "In version 1.0 of C-DNS [RFC8618], there is a field describing attributes of the Responses recorded. The field is a CBOR [RFC7049] unsigned integer holding bit flags."

The initial contents of the registry are as follows. See Section 7.3.2.4.1 of this document:

Bit	Name	Description	Reference
0	from-cache	The Response came from cache.	RFC 8618
1-63	Unassigned		

12.4. AddressEvent Types

IANA has created a registry "C-DNS Address Event Types" of C-DNS AddressEvent types. The primary purpose of this registry is to provide unique identifiers of different types of C-DNS address events and so specify the contents of the optional companion field "ae-code" for each type.

The following note is included in this registry: "In version 1.0 of C-DNS [RFC8618], there is a field identifying types of the events related to client addresses. This field is a CBOR [RFC7049] unsigned integer. There is a related optional field "ae-code", which, if present, holds an additional CBOR unsigned integer giving additional information specific to the event type."

The initial contents of the registry are as follows. See Section 7.3.2.5 of this document:

Identifier	Event Type	ae-code Contents	Reference
0	TCP reset	None	RFC 8618
1	ICMP time exceeded	ICMP code [icmpcodes]	RFC 8618
2	ICMP destination unreachable	ICMP code [icmpcodes]	RFC 8618
3	ICMPv6 time exceeded	ICMPv6 code [icmp6codes]	RFC 8618
4	ICMPv6 destination unreachable	ICMPv6 code [icmp6codes]	RFC 8618
5	ICMPv6 packet too big	ICMPv6 code [icmp6codes]	RFC 8618
6-18446744073709551615	Unassigned		

Expert reviewers should take the following point into consideration: "ae-code" contents must be defined for a type or, if not appropriate, specified as "None". A specification of "None" requires less storage and is therefore preferred.

13. Security Considerations

Any control interface MUST perform authentication and encryption.

Any data upload MUST be authenticated and encrypted.

14. Privacy Considerations

Storage of DNS traffic by operators in PCAP and other formats is a long-standing and widespread practice. Section 2.5 of [DNS-Priv-Cons] provides an analysis of the risks to Internet users regarding the storage of DNS traffic data in servers (recursive resolvers, authoritative servers, and rogue servers).

Section 5.2 of [DNS-Priv-Svc] describes mitigations for those risks for data stored on recursive resolvers (but that could by extension apply to authoritative servers). These include data-handling practices and methods for data minimization, IP address pseudonymization, and anonymization. Appendix C of [DNS-Priv-Svc] presents an analysis of seven published anonymization processes. In addition, the ICANN Root Server System Advisory Committee (RSSAC) have recently published [RSSAC04] ("Recommendations on Anonymization Processes for Source IP Addresses Submitted for Future Analysis").

The above analyses consider full data capture (e.g., using PCAP) as a baseline for privacy considerations; therefore, this format specification introduces no new user privacy issues beyond those of full data capture (which are quite severe). It does provide mechanisms to selectively record only certain fields at the time of data capture, to improve user privacy and to explicitly indicate that data is sampled, anonymized, or both. It also provides flags to indicate if data normalization has been performed; data normalization increases user privacy by reducing the potential for fingerprinting individuals. However, a trade-off is the potential reduction of the capacity to identify attack traffic via Query name signatures. Operators should carefully consider their operational requirements and privacy policies and SHOULD capture at the source the minimum user data required to meet their needs.

15. References

15.1. Normative References

[pcap-filter]
tcpdump.org, "Manpage of PCAP-FILTER", November 2017, <https://www.tcpdump.org/manpages/pcap-filter.7.html>.

[pcap-options]
tcpdump.org, "Manpage of PCAP", July 2018, <https://www.tcpdump.org/manpages/pcap.3pcap.html>.

[posix-time]
The Open Group, "IEEE Standard for Information Technology--Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7", IEEE Standard 1003.1-2017, Section 4.16, DOI 10.1109/IEEESTD.2018.8277153.

[RFC792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <https://www.rfc-editor.org/info/rfc792>.

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/info/rfc4443>.

[RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <https://www.rfc-editor.org/info/rfc6891>.

[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>.

[RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <https://www.rfc-editor.org/info/rfc7858>.

[RFC8094] Reddy, T., Wing, D., and P. Patil, "DNS over Datagram Transport Layer Security (DTLS)", RFC 8094, DOI 10.17487/RFC8094, February 2017, <https://www.rfc-editor.org/info/rfc8094>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <https://www.rfc-editor.org/info/rfc8484>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

15.2. Informative References

[Avro] The Apache Software Foundation, "Apache Avro(TM)", 2019, <https://avro.apache.org/>.

[ditl] DNS-OARC, "DITL", 2018, <https://www.dns-oarc.net/oarc/data/ditl>.

[DNS-Priv-Cons] Bortzmeyer, S. and S. Dickinson, "DNS Privacy Considerations", Work in Progress, draft-ietf-dprive-rfc7626-bis-00, July 2019.

[DNS-Priv-Svc] Dickinson, S., Overeinder, B., van Rijswijk-Deij, R., and A. Mankin, "Recommendations for DNS Privacy Service Operators", Work in Progress, draft-ietf-dprive-bcp-op-03, July 2019.

[dnscap] DNS-OARC, "DNSCAP", 2018, <https://www.dns-oarc.net/tools/dnscap>.

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

[dnstap-schema] "dnstap schema", commit d860ec1, November 2016, <https://github.com/dnstap/dnstap.pb/blob/master/dnstap.proto>.

[dnsxml] Daley, J., Ed., Morris, S., and J. Dickinson, "dnsxml - A standard XML representation of DNS data", Work in Progress, draft-daley-dnsxml-00, July 2013.

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

[gzip] "gzip", <https://www.gzip.org/>.

[icmp6codes]
IANA, "ICMPv6 "Code" Fields", <https://www.iana.org/assignments/icmpv6-parameters/>.

[icmpcodes]
IANA, "Code Fields", <https://www.iana.org/assignments/icmp-parameters/>.

[IEEE802.1Q]
IEEE, "IEEE Standard for Local and Metropolitan Area Networks--Bridges and Bridged Networks", IEEE Standard 802.1Q.

[Knot] "Knot DNS", <https://www.knot-dns.cz/>.

[lz4] "LZ4", <https://lz4.github.io/lz4/>.

[mmark] Gieben, M., "mmark", commit de69698, May 2019, <https://github.com/mmarkdown/mmark>.

[NSD] NLnet Labs, "NSD", 2019, <https://www.nlnetlabs.nl/projects/nsd/about/>.

[opcodes] IANA, "DNS OpCodes", <https://www.iana.org/assignments/dns-parameters/>.

[packetq] .SE - The Internet Infrastructure Foundation, "PacketQ", commit c9b2e89, February 2019, <https://github.com/DNS-OARC/PacketQ>.

[pcap] "PCAP", 2019, <https://www.tcpdump.org/>.

[pcapng] "pcapng: PCAP next generation file format specification", commit 3c35b6a, March 2019, <https://github.com/pcapng/pcapng>.

[Protocol-Buffers]
Google LLC, "Protocol Buffers", <https://developers.google.com/protocol-buffers/>.

[rcodes] IANA, "DNS RCODEs", <https://www.iana.org/assignments/dns-parameters/>.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <https://www.rfc-editor.org/info/rfc5077>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8427] Hoffman, P., "Representing DNS Messages in JSON", RFC 8427, DOI 10.17487/RFC8427, July 2018, <https://www.rfc-editor.org/info/rfc8427>.

[rrclasses] IANA, "DNS CLASSes", <https://www.iana.org/assignments/dns-parameters/>.

[rrtypes] IANA, "Resource Record (RR) TYPEs", <https://www.iana.org/assignments/dns-parameters/>.

[RSSAC04] ICANN, "Recommendations on Anonymization Processes for Source IP Addresses Submitted for Future Analysis", August 2018, <https://www.icann.org/en/system/files/files/rssac-040-07aug18-en.pdf>.

[snappy] "snappy", <https://google.github.io/snappy/>.

[snzip] "Snzip, a compression/decompression tool based on snappy", commit 809c6f2, October 2018, <https://github.com/kubo/snzip>.

[xz] "XZ Utils", <https://tukaani.org/xz/>.

[zstd] "Zstandard - Real-time data compression algorithm", <https://facebook.github.io/zstd/>.

Appendix A. CDDL

This appendix gives a CDDL [RFC8610] specification for C-DNS.

CDDL does not permit a range of allowed values to be specified for a bitfield. Where necessary, those values are given as a CDDL group, but the group definition is commented out to prevent CDDL tooling from warning that the group is unused.

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

;
; The overall structure of a file.
;
File = [
    file-type-id  : "C-DNS",
    file-preamble : FilePreamble,
    file-blocks   : [* Block],
]

;
; The File Preamble.
;
FilePreamble = {
    major-format-version => 1,
    minor-format-version => 0,
    ? private-version    => uint,
    block-parameters     => [+ BlockParameters],
}
major-format-version = 0
minor-format-version = 1
private-version      = 2
block-parameters     = 3

BlockParameters = {
    storage-parameters      => StorageParameters,
    ? collection-parameters => CollectionParameters,
}
storage-parameters    = 0
collection-parameters = 1

  IPv6PrefixLength = 1..128
  IPv4PrefixLength = 1..32
  OpcodeRange = 0..15
  RRTypeRange = 0..65535
```

```
     StorageParameters = {
         ticks-per-second             => uint,
         max-block-items              => uint,
         storage-hints                => StorageHints,
         opcodes                      => [+ OpcodeRange],
         rr-types                     => [+ RRTypeRange],
         ? storage-flags              => StorageFlags,
         ? client-address-prefix-ipv4 => IPv4PrefixLength,
         ? client-address-prefix-ipv6 => IPv6PrefixLength,
         ? server-address-prefix-ipv4 => IPv4PrefixLength,
         ? server-address-prefix-ipv6 => IPv6PrefixLength,
         ? sampling-method            => tstr,
         ? anonymization-method       => tstr,
     }
     ticks-per-second           = 0
     max-block-items            = 1
     storage-hints              = 2
     opcodes                    = 3
     rr-types                   = 4
     storage-flags              = 5
     client-address-prefix-ipv4 = 6
     client-address-prefix-ipv6 = 7
     server-address-prefix-ipv4 = 8
     server-address-prefix-ipv6 = 9
     sampling-method            = 10
     anonymization-method       = 11

       ; A hint indicates whether the collection method will always omit
       ; the item from the file.
       StorageHints = {
           query-response-hints           => QueryResponseHints,
           query-response-signature-hints =>
               QueryResponseSignatureHints,
           rr-hints                       => RRHints,
           other-data-hints               => OtherDataHints,
       }
       query-response-hints           = 0
       query-response-signature-hints = 1
       rr-hints                       = 2
       other-data-hints               = 3

         QueryResponseHintValues = &(
             time-offset                 : 0,
             client-address-index        : 1,
             client-port                 : 2,
             transaction-id              : 3,
             qr-signature-index          : 4,
             client-hoplimit             : 5,
```

```
        response-delay               : 6,
        query-name-index             : 7,
        query-size                   : 8,
        response-size                : 9,
        response-processing-data     : 10,
        query-question-sections      : 11,    ; Second & subsequent
                                              ; Questions
        query-answer-sections        : 12,
        query-authority-sections     : 13,
        query-additional-sections    : 14,
        response-answer-sections     : 15,
        response-authority-sections  : 16,
        response-additional-sections : 17,
    )
    QueryResponseHints = uint .bits QueryResponseHintValues

    QueryResponseSignatureHintValues = &(
        server-address-index  : 0,
        server-port           : 1,
        qr-transport-flags    : 2,
        qr-type               : 3,
        qr-sig-flags          : 4,
        query-opcode          : 5,
        qr-dns-flags          : 6,
        query-rcode           : 7,
        query-classtype-index : 8,
        query-qdcount         : 9,
        query-ancount         : 10,
        query-nscount         : 11,
        query-arcount         : 12,
        query-edns-version    : 13,
        query-udp-size        : 14,
        query-opt-rdata-index : 15,
        response-rcode        : 16,
    )
    QueryResponseSignatureHints =
        uint .bits QueryResponseSignatureHintValues

    RRHintValues = &(
        ttl         : 0,
        rdata-index : 1,
    )
    RRHints = uint .bits RRHintValues

    OtherDataHintValues = &(
        malformed-messages   : 0,
        address-event-counts : 1,
    )
```

```
         OtherDataHints = uint .bits OtherDataHintValues

       StorageFlagValues = &(
           anonymized-data      : 0,
           sampled-data         : 1,
           normalized-names     : 2,
       )
       StorageFlags = uint .bits StorageFlagValues

   ; Metadata about data collection
   VLANIdRange = 1..4094

   CollectionParameters = {
        ? query-timeout      => uint,             ; Milliseconds
        ? skew-timeout       => uint,             ; Microseconds
        ? snaplen            => uint,
        ? promisc            => bool,
        ? interfaces         => [+ tstr],
        ? server-addresses   => [+ IPAddress],
        ? vlan-ids           => [+ VLANIdRange],
        ? filter             => tstr,
        ? generator-id       => tstr,
        ? host-id            => tstr,
    }
    query-timeout      = 0
    skew-timeout       = 1
    snaplen            = 2
    promisc            = 3
    interfaces         = 4
    server-addresses   = 5
    vlan-ids           = 6
    filter             = 7
    generator-id       = 8
    host-id            = 9

  ;
  ; Data in the file is stored in Blocks.
  ;
  Block = {
      block-preamble          => BlockPreamble,
      ? block-statistics      => BlockStatistics, ; Much of this
                                                  ; could be derived
      ? block-tables          => BlockTables,
      ? query-responses       => [+ QueryResponse],
      ? address-event-counts  => [+ AddressEventCount],
      ? malformed-messages    => [+ MalformedMessage],
  }
```

```
   block-preamble        = 0
   block-statistics      = 1
   block-tables          = 2
   query-responses       = 3
   address-event-counts  = 4
   malformed-messages    = 5

   ;
   ; The (mandatory) preamble to a Block.
   ;
   BlockPreamble = {
       ? earliest-time          => Timestamp,
       ? block-parameters-index => uint .default 0,
   }
   earliest-time          = 0
   block-parameters-index = 1

   ; Ticks are sub-second intervals.  The number of ticks in a second is
   ; file/block metadata.  Signed and unsigned tick types are defined.
   ticks = int
   uticks = uint

   Timestamp = [
       timestamp-secs   : uint,      ; POSIX time
       timestamp-ticks  : uticks,
   ]

   ;
   ; Statistics about the Block contents.
   ;
   BlockStatistics = {
       ? processed-messages  => uint,
       ? qr-data-items       => uint,
       ? unmatched-queries   => uint,
       ? unmatched-responses => uint,
       ? discarded-opcode    => uint,
       ? malformed-items     => uint,
   }
   processed-messages  = 0
   qr-data-items       = 1
   unmatched-queries   = 2
   unmatched-responses = 3
   discarded-opcode    = 4
   malformed-items     = 5
```

```
;
; Tables of common data referenced from records in a Block.
;
BlockTables = {
    ? ip-address             => [+ IPAddress],
    ? classtype              => [+ ClassType],
    ? name-rdata             => [+ bstr],    ; Holds both names
                                             ; and RDATA
    ? qr-sig                 => [+ QueryResponseSignature],
    ? QuestionTables,
    ? RRTables,
    ? malformed-message-data => [+ MalformedMessageData],
}
ip-address             = 0
classtype              = 1
name-rdata             = 2
qr-sig                 = 3
qlist                  = 4
qrr                    = 5
rrlist                 = 6
rr                     = 7
malformed-message-data = 8

IPv4Address = bstr .size (0..4)
IPv6Address = bstr .size (0..16)
IPAddress = IPv4Address / IPv6Address

ClassType = {
    type  => uint,
    class => uint,
}
type  = 0
class = 1

QueryResponseSignature = {
    ? server-address-index  => uint,
    ? server-port           => uint,
    ? qr-transport-flags    => QueryResponseTransportFlags,
    ? qr-type               => QueryResponseType,
    ? qr-sig-flags          => QueryResponseFlags,
    ? query-opcode          => uint,
    ? qr-dns-flags          => DNSFlags,
    ? query-rcode           => uint,
    ? query-classtype-index => uint,
    ? query-qdcount         => uint,
    ? query-ancount         => uint,
    ? query-nscount         => uint,
    ? query-arcount         => uint,
```

```
       ? query-edns-version    => uint,
       ? query-udp-size        => uint,
       ? query-opt-rdata-index => uint,
       ? response-rcode        => uint,
   }
   server-address-index  = 0
   server-port           = 1
   qr-transport-flags    = 2
   qr-type               = 3
   qr-sig-flags          = 4
   query-opcode          = 5
   qr-dns-flags          = 6
   query-rcode           = 7
   query-classtype-index = 8
   query-qdcount         = 9
   query-ancount         = 10
   query-nscount         = 11
   query-arcount         = 12
   query-edns-version    = 13
   query-udp-size        = 14
   query-opt-rdata-index = 15
   response-rcode        = 16

     ; Transport gives the values that may appear in bits 1..4 of
     ; TransportFlags.  There is currently no way to express this in
     ; CDDL, so Transport is unused.  To avoid confusion when used
     ; with CDDL tools, it is commented out.
     ;
     ; Transport = &(
     ;     udp               : 0,
     ;     tcp               : 1,
     ;     tls               : 2,
     ;     dtls              : 3,
     ;     https             : 4,
     ;     non-standard      : 15,
     ; )

     TransportFlagValues = &(
         ip-version         : 0,     ; 0=IPv4, 1=IPv6
     ) / (1..4)
     TransportFlags = uint .bits TransportFlagValues

     QueryResponseTransportFlagValues = &(
         query-trailingdata : 5,
     ) / TransportFlagValues
     QueryResponseTransportFlags =
         uint .bits QueryResponseTransportFlagValues
```

```
QueryResponseType = &(
    stub      : 0,
    client    : 1,
    resolver  : 2,
    auth      : 3,
    forwarder : 4,
    tool      : 5,
)

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

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
```

```
   QuestionTables = (
       qlist => [+ QuestionList],
       qrr   => [+ Question]
   )

     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

   RRTables = (
       rrlist => [+ RRList],
       rr     => [+ RR]
   )

     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
     }
     ; Other map key values already defined above.
     ttl         = 2
     rdata-index = 3

   MalformedMessageData = {
       ? server-address-index   => uint,
       ? server-port            => uint,
       ? mm-transport-flags     => TransportFlags,
       ? mm-payload             => bstr,
   }
   ; Other map key values already defined above.
   mm-transport-flags      = 2
   mm-payload              = 3
```

```
   ;
   ; A single Query/Response data item.
   ;
   QueryResponse = {
       ? time-offset              => uticks,     ; Time offset from
                                                 ; start of Block
       ? client-address-index     => uint,
       ? client-port              => uint,
       ? transaction-id           => uint,
       ? qr-signature-index       => uint,
       ? client-hoplimit          => uint,
       ? response-delay           => ticks,
       ? query-name-index         => uint,
       ? query-size               => uint,       ; DNS size of Query
       ? response-size            => uint,       ; DNS size of Response
       ? response-processing-data => ResponseProcessingData,
       ? query-extended           => QueryResponseExtended,
       ? response-extended        => QueryResponseExtended,
   }
   time-offset              = 0
   client-address-index     = 1
   client-port              = 2
   transaction-id           = 3
   qr-signature-index       = 4
   client-hoplimit          = 5
   response-delay           = 6
   query-name-index         = 7
   query-size               = 8
   response-size            = 9
   response-processing-data = 10
   query-extended           = 11
   response-extended        = 12

   ResponseProcessingData = {
       ? bailiwick-index  => uint,
       ? processing-flags => ResponseProcessingFlags,
   }
   bailiwick-index = 0
   processing-flags = 1

     ResponseProcessingFlagValues = &(
         from-cache : 0,
     )
     ResponseProcessingFlags = uint .bits ResponseProcessingFlagValues
```

```
   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

   ;
   ; Address event data.
   ;
   AddressEventCount = {
       ae-type              => &AddressEventType,
       ? ae-code            => uint,
       ae-address-index     => uint,
       ? ae-transport-flags => TransportFlags,
       ae-count             => uint,
   }
   ae-type            = 0
   ae-code            = 1
   ae-address-index   = 2
   ae-transport-flags = 3
   ae-count           = 4

   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,
   )

   ;
   ; Malformed messages.
   ;
   MalformedMessage = {
       ? time-offset           => uticks,   ; Time offset from
                                            ; start of Block
       ? client-address-index  => uint,
       ? client-port           => uint,
       ? message-data-index    => uint,
   }
   ; Other map key values already defined above.
   message-data-index = 3
```

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 (1) labels at the end of the name can be replaced with a reference back to part (or all) of the earlier name and (2) 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 step-by-step process of adding names and performing name compression. In an example packet, the first name added is foo.example, which cannot be compressed.

N	Name	Uncompressed	Compression Target
1	foo.example	foo.example	None

The next name added is bar.example. This is matched against foo.example. The example 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	foo.example	foo.example	None
2	bar.example	bar	1 + offset to example

The third name added is www.bar.example. This is first matched against foo.example, 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	foo.example	foo.example	None
2	bar.example	bar	1 + offset to example
3	www.bar.example	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 the Name Server Daemon (NSD) [NSD] 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 name server [Knot] 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 as follows:

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 than those produced by NSD.

Appendix C. Comparison of Binary Formats

Several binary serialization formats were considered. For completeness, they were also compared to JSON.

- Apache Avro [Avro]. Data is stored according to a predefined schema. The schema itself is always included in the data file. Data can therefore be stored untagged, for a smaller serialization 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 [Protocol-Buffers]. Data is stored according to a predefined 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 serialization size. The schema is not stored with the data, so unlike Avro, it cannot be read with a generic library.

  * Code must be generated for a particular data schema 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 [RFC7049]. This serialization format is comparable to JSON but with a binary representation. It does not use a predefined schema, so data is always stored tagged. However, CBOR data schemas can be described using CDDL [RFC8610], and tools exist to verify that data files conform to the schema.

  * CBOR is a simple format and is 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 that 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 [snzip]. A command-line compression tool based on the Google Snappy library [snappy].

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

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

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

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

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

Note that this document 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 662 MB 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 is in kB, and user time is 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 Media Access Control (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 deduplication on Query/Response records within the block. The simple and block formats shown 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 deduplication 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 deduplication 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 than 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 document anticipates that output data will be subject to compression. There is no compelling case for one particular binary serialization 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 serialization format for the reasons listed in Section 5.

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:

1. What is the impact on performance of using different block sizes in the format file?

2. 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 sizes are in MB, RSS is in kB, and user time is in seconds.

Block Size	File Size	RSS	User Time
1,000	133.46	612.27	15.25
5,000	89.85	676.82	14.99
10,000	76.87	752.40	14.53
20,000	67.86	750.75	14.49
40,000	61.88	736.30	14.29
80,000	58.08	694.16	14.28
160,000	55.94	733.84	14.44
320,000	54.41	799.20	13.97

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

The following table demonstrates the effect of increasing block size on output file size for different compressions.

Block Size	None	snzip	lz4	gzip	zstd	xz
1,000	133.46	90.52	90.03	74.65	44.78	25.63
5,000	89.85	59.69	59.43	46.99	37.33	22.34
10,000	76.87	50.39	50.28	38.94	33.62	21.09
20,000	67.86	43.91	43.90	33.24	32.62	20.16
40,000	61.88	39.63	39.69	29.44	28.72	19.52
80,000	58.08	36.93	37.01	27.05	26.25	19.00
160,000	55.94	35.10	35.06	25.44	24.56	19.63
320,000	54.41	33.87	33.74	24.36	23.44	18.66

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 scheme, block sizes over 10,000 Query/Response pairs would seem to offer limited improvements.

Appendix D. Data Fields for Traffic Regeneration

D.1. Recommended Fields for Traffic Regeneration

This section specifies the data fields that would need to be captured in order to perform the fullest PCAP traffic reconstruction for well-formed DNS messages that is possible with C-DNS.

- o All data fields in the QueryResponse type except response-processing-data.

- o All data fields in the QueryResponseSignature type except qr-type.

- o All data fields in the RR TYPE.

D.2. Issues with Small Data Captures

At the other extreme, an interesting corner case arises when opting to perform captures with a smaller data set than that recommended above. The following list specifies a subset of the above data fields; if only these data fields are captured, then even a minimal traffic reconstruction is problematic because there is not enough information to determine if the Query/Response data item contained just a Query, just a Response, or a Query/Response pair.

- The following data fields from the QueryResponse type:

  * time-offset

  * client-address-index

  * client-port

  * transaction-id

  * query-name-index

- The following data fields from the QueryResponseSignature type:

  * server-address-index

  * server-port

  * qr-transport-flags

  * query-classtype-index

In this case, simply also capturing the qr-sig-flags will provide enough information to perform a minimal traffic reconstruction (assuming that suitable defaults for the remaining fields are provided). Additionally, capturing response-delay, query-opcode, and response-rcode will avoid having to rely on potentially misleading defaults for these values and should result in a PCAP that represents the basics of the real traffic flow.

Acknowledgements

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

Thanks also to Robert Edmonds, Jerry Lundstrom, Richard Gibson, Stephane Bortzmeyer, and many other members of DNSOP for review.

Also, thanks to Miek Gieben for [mmark].

Authors' Addresses

John Dickinson
Sinodun IT
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA
United Kingdom
Email: jad@sinodun.com

Jim Hague
Sinodun IT
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA
United Kingdom
Email: jim@sinodun.com

Sara Dickinson
Sinodun IT
Magdalen Centre
Oxford Science Park
Oxford OX4 4GA
United Kingdom
Email: sara@sinodun.com

Terry Manderson
ICANN
12025 Waterfront Drive
Suite 300
Los Angeles, CA 90094-2536
United States of America
Email: terry.manderson@icann.org

John Bond
Wikimedia Foundation, Inc.
1 Montgomery Street
Suite 1600
San Francisco, CA 94104
United States of America
Email: ietf-wikimedia@johnbond.org