Stream:	Internet Engineering Task Force (IETF)
RFC:	9132
Obsoletes:	8782
Category:	Standards Track
Published:	September 2021
ISSN:	2070-1721
Authors:	M. Boucadair, Ed.	J. Shallow	T. Reddy.K
---	---	---	---
	Orange		*Akamai*

RFC 9132
Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification

Abstract

This document specifies the Distributed Denial-of-Service Open Threat Signaling (DOTS) signal channel, a protocol for signaling the need for protection against Distributed Denial-of-Service (DDoS) attacks to a server capable of enabling network traffic mitigation on behalf of the requesting client.

A companion document defines the DOTS data channel, a separate reliable communication layer for DOTS management and configuration purposes.

This document obsoletes RFC 8782.

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/rfc9132.

Table of Contents

1. Introduction

A Distributed Denial-of-Service (DDoS) attack is a distributed attempt to make machines or network resources unavailable to their intended users. In most cases, sufficient scale for an effective attack can be achieved by compromising enough end hosts and using those infected hosts to perpetrate and amplify the attack. The victim in this attack can be an application server, a host, a router, a firewall, or an entire network.

Network applications have finite resources, like CPU cycles, the number of processes or threads they can create and use, the maximum number of simultaneous connections they can handle, the resources assigned to the control plane, etc. When processing network traffic, such applications are supposed to use these resources to provide the intended functionality in the most efficient manner. However, a DDoS attacker may be able to prevent an application from performing its intended task by making the application exhaust its finite resources.

A TCP DDoS SYN flood [RFC4987], for example, is a memory-exhausting attack, while an ACK flood is a CPU-exhausting attack. Attacks on the link are carried out by sending enough traffic so that the link becomes congested, thereby likely causing packet loss for legitimate traffic. Stateful firewalls can also be attacked by sending traffic that causes the firewall to maintain an excessive number of states that may jeopardize the firewall's operation overall, in addition to likely performance impacts. The firewall then runs out of memory, and it can no longer instantiate the states required to process legitimate flows. Other possible DDoS attacks are discussed in [RFC4732].

In many cases, it may not be possible for network administrators to determine the cause(s) of an attack. They may instead just realize that certain resources seem to be under attack. This document defines a lightweight protocol that allows a DOTS client to request mitigation from one or more DOTS servers for protection against detected, suspected, or anticipated attacks. This protocol enables cooperation between DOTS agents to permit a highly automated network defense that is robust, reliable, and secure. Note that "secure" means the support of the features defined in Section 2.4 of [RFC8612].

In typical deployments, the DOTS client belongs to a different administrative domain than the DOTS server. For example, the DOTS client is embedded in a firewall-protected service owned and operated by a customer, while the DOTS server is owned and operated by a different administrative entity (service provider, typically) providing DDoS mitigation services. The latter might or might not provide connectivity services to the network hosting the DOTS client.

The DOTS server may or may not be co-located with the DOTS mitigator. In typical deployments, the DOTS server belongs to the same administrative domain as the mitigator. The DOTS client can communicate directly with a DOTS server or indirectly via a DOTS gateway.

An example of a network diagram that illustrates a deployment of DOTS agents is shown in Figure 1. In this example, a DOTS server is operating on the access network. A DOTS client is located on the Local Area Network (LAN), while a DOTS gateway is embedded in the Customer Premises Equipment (CPE).

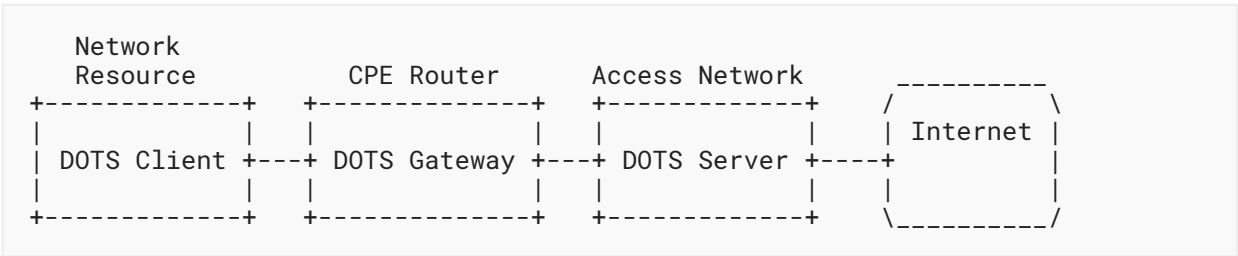

Figure 1: Sample DOTS Deployment (1)

DOTS servers can also be reachable over the Internet, as depicted in Figure 2.

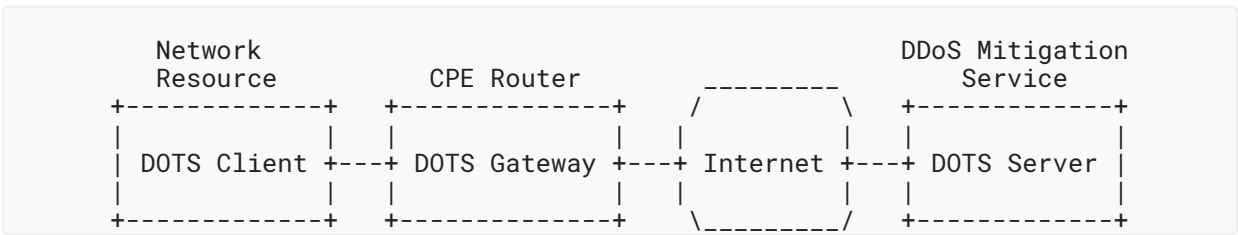

Figure 2: Sample DOTS Deployment (2)

This document adheres to the DOTS architecture [RFC8811]. The requirements for the DOTS signal channel protocol are documented in [RFC8612]. This document satisfies all the use cases discussed in [RFC8903].

This document focuses on the DOTS signal channel. This is a companion document of the DOTS data channel specification [RFC8783] that defines a configuration and a bulk data exchange mechanism supporting the DOTS signal channel.

Backward compatibility (including upgrade) considerations are discussed in Section 3.1.

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.

(D)TLS is used for statements that apply to both Transport Layer Security [RFC5246] [RFC8446] and Datagram Transport Layer Security [RFC6347]. Specific terms are used for any statement that applies to either protocol alone.

The reader should be familiar with the terms defined in [RFC8612] and [RFC7252].

The meaning of the symbols in YANG tree diagrams are defined in [RFC8340] and [RFC8791].

3. Design Overview

The DOTS signal channel is built on top of the Constrained Application Protocol (CoAP) [RFC7252], a lightweight protocol originally designed for constrained devices and networks. The many features of CoAP (expectation of packet loss, support for asynchronous Non-confirmable messaging, congestion control, small message overhead limiting the need for fragmentation, use of minimal resources, and support for (D)TLS) make it a good candidate upon which to build the DOTS signaling mechanism.

DOTS clients and servers behave as CoAP endpoints. By default, a DOTS client behaves as a CoAP client and a DOTS server behaves as CoAP server. Nevertheless, a DOTS client (or server) behaves as a CoAP server (or client) for specific operations, such as DOTS heartbeat operations (Section 4.7).

The DOTS signal channel is layered on existing standards (see Figure 3).

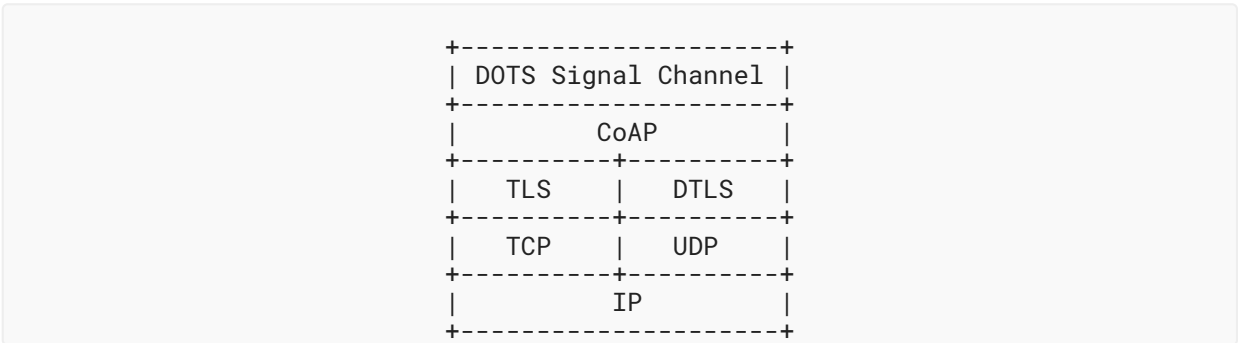

Figure 3: Abstract Layering of DOTS Signal Channel over CoAP over (D)TLS

In some cases, a DOTS client and server may have a mutual agreement to use a specific port number, such as by explicit configuration or dynamic discovery [RFC8973]. Absent such mutual agreement, the DOTS signal channel **MUST** run over port number 4646, as defined in Section 10.1, for both UDP and TCP (that is, the DOTS server listens on port number 4646). In order to use a distinct port number (as opposed to 4646), DOTS clients and servers **SHOULD** support a configurable parameter to supply the port number to use.

> Note: The rationale for not using the default port number 5684 ((D)TLS CoAP) is to avoid the discovery of services and resources discussed in [RFC7252] and allow for differentiated behaviors in environments where both a DOTS gateway and an Internet of Things (IoT) gateway (e.g., Figure 3 of [RFC7452]) are co-located.
>
> Particularly, the use of a default port number is meant to simplify DOTS deployment in scenarios where no explicit IP address configuration is required. For example, the use of the default router as the DOTS server aims to ease DOTS deployment within LANs (in which CPEs embed a DOTS gateway, as illustrated in Figures 1 and 2) without requiring a sophisticated discovery method and configuration tasks within the LAN. It is also possible to use anycast addresses for DOTS servers to simplify DOTS client configuration, including service discovery. In such an anycast-based scenario, a DOTS client initiating a DOTS session to the DOTS server anycast address may, for example, be (1) redirected to the DOTS server unicast address to be used by the DOTS client following the procedure discussed in Section 4.6 or (2) relayed to a unicast DOTS server.

The signal channel uses the "coaps" URI scheme defined in Section 6 of [RFC7252] and the "coaps+tcp" URI scheme defined in Section 8.2 of [RFC8323] to identify DOTS server resources that are accessible using CoAP over UDP secured with DTLS and CoAP over TCP secured with TLS, respectively.

The DOTS signal channel can be established between two DOTS agents prior to or during an attack. The DOTS signal channel is initiated by the DOTS client. The DOTS client can then negotiate, configure, and retrieve the DOTS signal channel session behavior with its DOTS peer (Section 4.5). Once the signal channel is established, the DOTS agents may periodically send heartbeats to keep the channel active (Section 4.7). At any time, the DOTS client may send a mitigation request message (Section 4.4) to a DOTS server over the active signal channel. While mitigation is active (because of the higher likelihood of packet loss during a DDoS attack), the DOTS server periodically sends status messages to the client, including basic mitigation feedback details. Mitigation remains active until the DOTS client explicitly terminates mitigation or the mitigation lifetime expires. Also, the DOTS server may rely on the signal channel session loss to trigger mitigation for preconfigured mitigation requests (if any).

DOTS signaling can use DTLS over UDP and TLS over TCP. Likewise, DOTS requests may be sent using IPv4 or IPv6 transfer capabilities. A Happy Eyeballs procedure for the DOTS signal channel is specified in Section 4.3.

A DOTS client is entitled to access only the resources it creates. In particular, a DOTS client cannot retrieve data related to mitigation requests created by other DOTS clients of the same DOTS client domain.

Messages exchanged between DOTS agents are serialized using Concise Binary Object Representation (CBOR) [RFC8949], a binary encoding scheme designed for small code and message size. CBOR-encoded payloads are used to carry signal-channel-specific payload messages that convey request parameters and response information, such as errors. In order to allow the reusing of data models across protocols, [RFC7951] specifies the JavaScript Object Notation (JSON) encoding of YANG-modeled data. A similar effort for CBOR is defined in [CORE-YANG-CBOR].

DOTS agents determine that a CBOR data structure is a DOTS signal channel object from the application context, such as from the port number assigned to the DOTS signal channel. The other method DOTS agents use to indicate that a CBOR data structure is a DOTS signal channel object is the use of the "application/dots+cbor" content type (Section 10.3).

This document specifies a YANG module for representing DOTS mitigation scopes, DOTS signal channel session configuration data, and DOTS redirected signaling (Section 5). All parameters in the payload of the DOTS signal channel are mapped to CBOR types, as specified in Table 5 (Section 6).

In order to prevent fragmentation, DOTS agents must follow the recommendations documented in Section 4.6 of [RFC7252]. Refer to Section 7.3 for more details.

DOTS agents **MUST** support GET, PUT, and DELETE CoAP methods. The payload included in CoAP responses with 2.xx Response Codes **MUST** be of content type "application/dots+cbor". CoAP responses with 4.xx and 5.xx error Response Codes **MUST** include a diagnostic payload (Section 5.5.2 of [RFC7252]). The diagnostic payload may contain additional information to aid troubleshooting.

In deployments where multiple DOTS clients are enabled in a single network and administrative domain (called DOTS client domain), the DOTS server may detect conflicting mitigation requests from these clients. This document does not aim to specify a comprehensive list of conditions under which a DOTS server will characterize two mitigation requests from distinct DOTS clients as conflicting, nor does it recommend a DOTS server behavior for processing conflicting mitigation requests. Those considerations are implementation and deployment specific. Nevertheless, this document specifies the mechanisms to notify DOTS clients when conflicts occur, including the conflict cause (Section 4.4.1.3).

In deployments where one or more translators (e.g., Traditional NAT [RFC3022], CGN [RFC6888], NAT64 [RFC6146], NPTv6 [RFC6296]) are enabled between the client's network and the DOTS server, any DOTS signal channel messages forwarded to a DOTS server **MUST NOT** include internal IP addresses/prefixes and/or port numbers; instead, external addresses/prefixes and/or

port numbers as assigned by the translator **MUST** be used. This document does not make any recommendations about possible translator discovery mechanisms. The following are some (non-exhaustive) deployment examples that may be considered:

- Port Control Protocol (PCP) [RFC6887] or Session Traversal Utilities for NAT (STUN) [RFC8489] may be used by the client to retrieve the external addresses/prefixes and/or port numbers. Information retrieved by means of PCP or STUN will be used to feed the DOTS signal channel messages that will be sent to a DOTS server.
- A DOTS gateway may be co-located with the translator. The DOTS gateway will need to update the DOTS messages based upon the local translator's binding table.

3.1. Backward Compatibility Considerations

The main changes to [RFC8782] are listed in Appendix A. These modifications are made with the constraint to avoid changes to the mapping table defined in Table 5 of [RFC8782] (see also Section 6 of the present document).

For both legacy [RFC8782] and implementations that follow the present specification, a DOTS signal channel attribute will thus have the same CBOR key value and CBOR major type. The only upgrade that is required to [RFC8782] implementations is to handle the CBOR key value range "128-255" as comprehension-optional instead of comprehension-required. Note that this range is anticipated to be used by the DOTS telemetry [DOTS-TELEMETRY] in which the following means are used to prevent message processing failure of a DOTS signal channel message enriched with telemetry data: (1) DOTS agents use dedicated (new) path suffixes (Section 5 of [DOTS-TELEMETRY]) and (2) a DOTS server won't include telemetry attributes in its responses unless it is explicitly told to do so by a DOTS client (Section 6.1.2 of [DOTS-TELEMETRY]).

Future DOTS extensions that request a CBOR value in the range "128-255" **MUST** support means similar to the aforementioned DOTS telemetry ones.

4. DOTS Signal Channel: Messages & Behaviors

4.1. DOTS Server(s) Discovery

This document assumes that DOTS clients are provisioned with the reachability information of their DOTS server(s) using any of a variety of means (e.g., local configuration or dynamic means, such as DHCP [RFC8973]). The description of such means is out of scope of this document.

Likewise, it is out of the scope of this document to specify the behavior to be followed by a DOTS client in order to send DOTS requests when multiple DOTS servers are provisioned (e.g., contact all DOTS servers, select one DOTS server among the list). Such behavior is specified in other documents (e.g., [DOTS-MULTIHOMING]).

4.2. CoAP URIs

The DOTS server **MUST** support the use of the path prefix of "/.well-known/" as defined in [RFC8615] and the registered name of "dots". Each DOTS operation is denoted by a path suffix that indicates the intended operation. The operation path (Table 1) is appended to the path prefix to form the URI used with a CoAP request to perform the desired DOTS operation.

Operation	Operation Path	Details
Mitigation	/mitigate	Section 4.4
Session configuration	/config	Section 4.5
Heartbeat	/hb	Section 4.7

Table 1: Operations and Corresponding URIs

4.3. Happy Eyeballs for DOTS Signal Channel

[RFC8612] mentions that DOTS agents will have to support both connectionless and connection-oriented protocols. As such, the DOTS signal channel is designed to operate with DTLS over UDP and TLS over TCP. Further, a DOTS client may acquire a list of IPv4 and IPv6 addresses (Section 4.1), each of which can be used to contact the DOTS server using UDP and TCP. If no list of IPv4 and IPv6 addresses to contact the DOTS server is configured (or discovered), the DOTS client adds the IPv4/IPv6 addresses of its default router to the candidate list to contact the DOTS server.

The following specifies the procedure to follow to select the address family and the transport protocol for sending DOTS signal channel messages.

Such a procedure is needed to avoid experiencing long connection delays. For example, if an IPv4 path to a DOTS server is functional, but the DOTS server's IPv6 path is nonfunctional, a dual-stack DOTS client may experience a significant connection delay compared to an IPv4-only DOTS client in the same network conditions. The other problem is that if a middlebox between the DOTS client and DOTS server is configured to block UDP traffic, the DOTS client will fail to establish a DTLS association with the DOTS server; consequently, it will have to fall back to TLS over TCP, thereby incurring significant connection delays.

To overcome these connection setup problems, the DOTS client attempts to connect to its DOTS server(s) using both IPv6 and IPv4, and it tries both DTLS over UDP and TLS over TCP following a DOTS Happy Eyeballs approach. To some extent, this approach is similar to the Happy Eyeballs mechanism defined in [RFC8305]. The connection attempts are performed by the DOTS client when it initializes or, in general, when it has to select an address family and transport to contact

its DOTS server. The results of the Happy Eyeballs procedure are used by the DOTS client for sending its subsequent messages to the DOTS server. The differences in behavior with respect to the Happy Eyeballs mechanism [RFC8305] are listed below:

- The order of preference of the DOTS signal channel address family and transport protocol (most preferred first) is the following: UDP over IPv6, UDP over IPv4, TCP over IPv6, and finally TCP over IPv4. This order adheres to the address preference order specified in [RFC6724] and the DOTS signal channel preference that promotes the use of UDP over TCP (to avoid TCP's head of line blocking).
- After successfully establishing a connection, the DOTS client **MUST** cache information regarding the outcome of each connection attempt for a specific time period; it uses that information to avoid thrashing the network with subsequent attempts. The cached information is flushed when its age exceeds a specific time period on the order of few minutes (e.g., 10 min). Typically, if the DOTS client has to reestablish the connection with the same DOTS server within a few seconds after the Happy Eyeballs mechanism is completed, caching avoids thrashing the network especially in the presence of DDoS attack traffic.
- If a DOTS signal channel session is established with TLS (but DTLS failed), the DOTS client periodically repeats the mechanism to discover whether DOTS signal channel messages with DTLS over UDP become available from the DOTS server; this is so the DOTS client can migrate the DOTS signal channel from TCP to UDP. Such probing **SHOULD NOT** be done more frequently than every 24 hours and **MUST NOT** be done more frequently than every 5 minutes.

When connection attempts are made during an attack, the DOTS client **SHOULD** use a "Connection Attempt Delay" [RFC8305] set to 100 ms.

In Figure 4, the DOTS client proceeds with the connection attempts following the rules in [RFC8305]. In this example, it is assumed that the IPv6 path is broken and UDP traffic is dropped by a middlebox, but this has little impact on the DOTS client because there is not a long delay before using IPv4 and TCP.

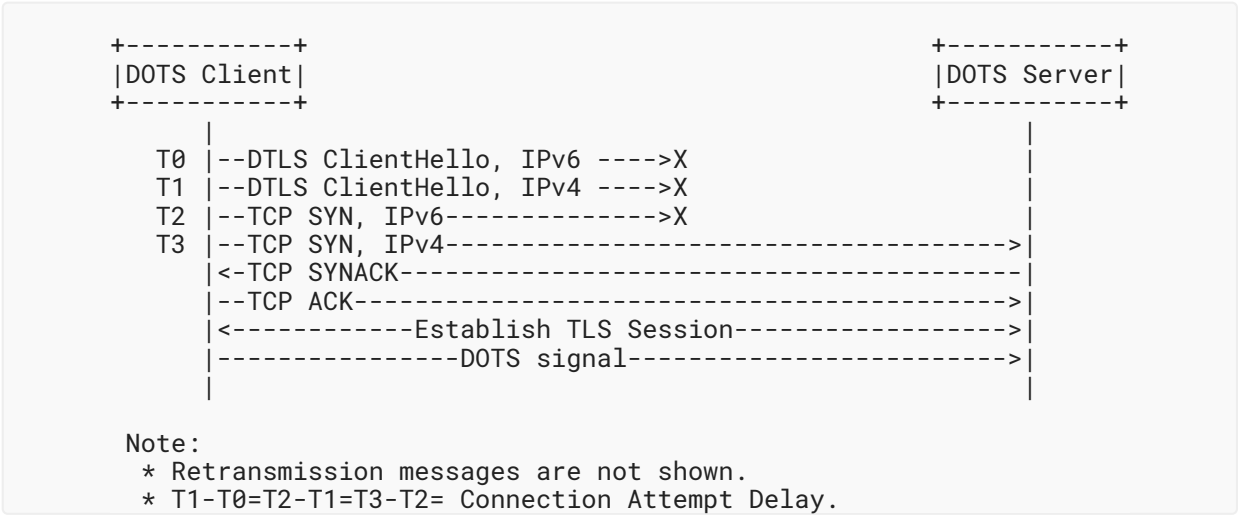

Figure 4: DOTS Happy Eyeballs (Sample Flow)

A single DOTS signal channel between DOTS agents can be used to exchange multiple DOTS signal messages. To reduce DOTS client and DOTS server workload, DOTS clients **SHOULD** reuse the (D)TLS session.

4.4. DOTS Mitigation Methods

The following methods are used by a DOTS client to request, retrieve, or withdraw the status of mitigation requests:

PUT: DOTS clients use the PUT method to request mitigation from a DOTS server (Section 4.4.1). During active mitigation, DOTS clients may use PUT requests to carry mitigation efficacy updates to the DOTS server (Section 4.4.3).

GET: DOTS clients may use the GET method to retrieve the list of its mitigations maintained by a DOTS server (Section 4.4.2) or to receive asynchronous DOTS server status messages (Section 4.4.2.1).

DELETE: DOTS clients use the DELETE method to withdraw a request for mitigation from a DOTS server (Section 4.4.4).

Mitigation request and response messages are marked as Non-confirmable messages (Section 2.2 of [RFC7252]).

DOTS agents **MUST NOT** send more than one UDP datagram per round-trip time (RTT) to the peer DOTS agent on average following the data transmission guidelines discussed in Section 3.1.3 of [RFC8085].

Requests marked by the DOTS client as Non-confirmable messages are sent at regular intervals until a response is received from the DOTS server. If the DOTS client cannot maintain an RTT estimate, it **MUST NOT** send more than one Non-confirmable request every 3 seconds and

SHOULD use an even less aggressive rate whenever possible (case 2 in Section 3.1.3 of [RFC8085]). Mitigation requests **MUST NOT** be delayed because of checks on probing rate (Section 4.7 of [RFC7252]).

JSON encoding of YANG-modeled data [RFC7951] is used to illustrate the various methods defined in the following subsections. Also, the examples use the Labels defined in Sections 10.6.2, 10.6.3, 10.6.4, and 10.6.5.

The DOTS client **MUST** authenticate itself to the DOTS server (Section 8). The DOTS server **MAY** use the algorithm presented in Section 7 of [RFC7589] to derive the DOTS client identity or username from the client certificate. The DOTS client identity allows the DOTS server to accept mitigation requests with scopes that the DOTS client is authorized to manage.

4.4.1. Request Mitigation

4.4.1.1. Building Mitigation Requests

When a DOTS client requires mitigation for some reason, the DOTS client uses the CoAP PUT method to send a mitigation request to its DOTS server(s) (Figures 5 and 6).

If a DOTS client is entitled to solicit the DOTS service, the DOTS server enables mitigation on behalf of the DOTS client by communicating the DOTS client's request to a mitigator (which may be co-located with the DOTS server) and relaying the feedback of the thus-selected mitigator to the requesting DOTS client.

```
Header: PUT (Code=0.03)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "mitigate"
Uri-Path: "cuid=dz6pHjaADkaFTbjr0JGBpw"
Uri-Path: "mid=123"
Content-Format: "application/dots+cbor"

{
  ...
}
```

Figure 5: PUT to Convey DOTS Mitigation Requests

The order of the Uri-Path options is important, as it defines the CoAP resource. In particular, 'mid' **MUST** follow 'cuid'.

The additional Uri-Path parameters to those defined in Section 4.2 are as follows:

cuid: Stands for Client Unique Identifier. A globally unique identifier that is meant to prevent collisions among DOTS clients, especially those from the same domain. It **MUST** be generated by DOTS clients.

For the reasons discussed in Appendix B, implementations **SHOULD** set 'cuid' using the following procedure: first, the DOTS client inputs one of the following into the SHA-256 [RFC6234] cryptographic hash: the DER-encoded ASN.1 representation of the Subject Public Key Info (SPKI) of its X.509 certificate [RFC5280], its raw public key [RFC7250], the "Pre-Shared Key (PSK) identity" it uses in the TLS 1.2 ClientKeyExchange message, or the "identity" it uses in the "pre_shared_key" TLS 1.3 extension. Then, the output of the cryptographic hash algorithm is truncated to 16 bytes; truncation is done by stripping off the final 16 bytes. The truncated output is base64url encoded (Section 5 of [RFC4648]) with the two trailing "=" removed from the encoding, and the resulting value used as the 'cuid'.

The 'cuid' is intended to be stable when communicating with a given DOTS server, i.e., the 'cuid' used by a DOTS client **SHOULD NOT** change over time. Distinct 'cuid' values **MAY** be used by a single DOTS client per DOTS server.

If a DOTS client has to change its 'cuid' for some reason, it **MUST NOT** do so when mitigations are still active for the old 'cuid'. The 'cuid' **SHOULD** be 22 characters to avoid DOTS signal message fragmentation over UDP. Furthermore, if that DOTS client created aliases and filtering entries at the DOTS server by means of the DOTS data channel, it **MUST** delete all the entries bound to the old 'cuid' and reinstall them using the new 'cuid'.

DOTS servers **MUST** return 4.09 (Conflict) error code to a DOTS peer to notify that the 'cuid' is already in use by another DOTS client. Upon receipt of that error code, a new 'cuid' **MUST** be generated by the DOTS peer (e.g., using [RFC4122]).

Client-domain DOTS gateways **MUST** handle 'cuid' collision directly, and it is **RECOMMENDED** that 'cuid' collision is handled directly by server-domain DOTS gateways.

DOTS gateways **MAY** rewrite the 'cuid' used by peer DOTS clients. Triggers for such rewriting are out of scope.

This is a mandatory Uri-Path parameter.

mid: Identifier for the mitigation request represented with an integer. This identifier **MUST** be unique for each mitigation request bound to the DOTS client, i.e., the 'mid' parameter value in the mitigation request needs to be unique (per 'cuid' and DOTS server) relative to the 'mid' parameter values of active mitigation requests conveyed from the DOTS client to the DOTS server.

In order to handle out-of-order delivery of mitigation requests, 'mid' values **MUST** increase monotonically.

If the 'mid' value has reached 3/4 of $(2^{(32)} - 1)$ (i.e., 3221225471) and no attack is detected, the DOTS client **MUST** reset 'mid' to 0 to handle 'mid' rollover. If the DOTS client maintains mitigation requests with preconfigured scopes, it **MUST** recreate them with the 'mid' restarting at 0.

This identifier **MUST** be generated by the DOTS client.

This is a mandatory Uri-Path parameter.

'cuid' and 'mid' **MUST NOT** appear in the PUT request message body (Figure 6). The schema in Figure 6 uses the types defined in Section 6. Note that this figure (and other similar figures depicting a schema) are non-normative sketches of the structure of the message.

```
{
  "ietf-dots-signal-channel:mitigation-scope": {
    "scope": [
      {
        "target-prefix": [
          "string"
        ],
        "target-port-range": [
          {
            "lower-port": number,
            "upper-port": number
          }
        ],
        "target-protocol": [
          number
        ],
        "target-fqdn": [
          "string"
        ],
        "target-uri": [
          "string"
        ],
        "alias-name": [
          "string"
        ],
        "lifetime": number,
        "trigger-mitigation": true|false
      }
    ]
  }
}
```

Figure 6: PUT to Convey DOTS Mitigation Requests (Message Body Schema)

The parameters in the CBOR body (Figure 6) of the PUT request are described below:

target-prefix: A list of prefixes identifying resources under attack. Prefixes are represented using Classless Inter-Domain Routing (CIDR) notation [RFC4632].

The prefix length must be less than or equal to 32 for IPv4 and 128 for IPv6.

The prefix list **MUST NOT** include broadcast, loopback, or multicast addresses. These addresses are considered to be invalid values. In addition, the DOTS server **MUST** validate that target prefixes are within the scope of the DOTS client domain. Other validation checks may be supported by DOTS servers.

This is an optional attribute.

target-port-range: A list of port numbers bound to resources under attack.

A port range is defined by two bounds: a lower port number ('lower-port') and an upper port number ('upper-port'). When only 'lower-port' is present, it represents a single port number.

For TCP, UDP, Stream Control Transmission Protocol (SCTP) [RFC4960], or Datagram Congestion Control Protocol (DCCP) [RFC4340], a range of ports can be, for example, 0-1023, 1024-65535, or 1024-49151.

This is an optional attribute.

target-protocol: A list of protocols involved in an attack. Values are integers in the range of 0 to 255. See [IANA-Proto] for common values.

If 'target-protocol' is not specified, then the request applies to any protocol.

This is an optional attribute.

target-fqdn: A list of Fully Qualified Domain Names (FQDNs) identifying resources under attack [RFC8499].

How a name is passed to an underlying name resolution library is implementation and deployment specific. Nevertheless, once the name is resolved into one or multiple IP addresses, DOTS servers **MUST** apply the same validation checks as those for 'target-prefix'. These validation checks are reiterated by DOTS servers each time a name is passed to an underlying name resolution library (e.g., upon expiry of DNS TTL).

The use of FQDNs may be suboptimal because:

- It induces both an extra load and increased delays on the DOTS server to handle and manage DNS resolution requests.
- It does not guarantee that the DOTS server will resolve a name to the same IP addresses that the DOTS client does.

This is an optional attribute.

target-uri: A list of URIs [RFC3986] identifying resources under attack.

The same validation checks used for 'target-fqdn' **MUST** be followed by DOTS servers to validate a target URI.

This is an optional attribute.

alias-name: A list of aliases of resources for which the mitigation is requested. Aliases can be created using the DOTS data channel (Section 6.1 of [RFC8783]), direct configuration, or other means.

An alias is used in subsequent signal channel exchanges to refer more efficiently to the resources under attack.

This is an optional attribute.

lifetime: Lifetime of the mitigation request in seconds. The **RECOMMENDED** lifetime of a mitigation request is 3600 seconds; this value was chosen to be long enough so that refreshing is not typically a burden on the DOTS client while still making the request expire in a timely manner when the client has unexpectedly quit. DOTS clients **MUST** include this parameter in their mitigation requests.

A lifetime of '0' in a mitigation request is an invalid value.

A lifetime of negative one (-1) indicates indefinite lifetime for the mitigation request. The DOTS server **MAY** refuse an indefinite lifetime, for policy reasons; the granted lifetime value is returned in the response. DOTS clients **MUST** be prepared to not be granted mitigations with indefinite lifetimes.

The DOTS server **MUST** always indicate the actual lifetime in the response and the remaining lifetime in status messages sent to the DOTS client.

Upon the expiry of the negotiated lifetime (i.e., the remaining lifetime reaches '0'), and if the request is not refreshed by the DOTS client, the mitigation request is removed by the DOTS server. The request can be refreshed by sending the same request again.

This is a mandatory attribute.

trigger-mitigation: If the parameter value is set to 'false', DDoS mitigation will not be triggered for the mitigation request unless the DOTS signal channel session is lost.

If the DOTS client ceases to respond to heartbeat messages, the DOTS server can detect that the DOTS signal channel session is lost. More details are discussed in Section 4.7.

The default value of the parameter is 'true' (that is, the mitigation starts immediately). If 'trigger-mitigation' is not present in a request, this is equivalent to receiving a request with 'trigger-mitigation' set to 'true'.

This is an optional attribute.

Because of the complexity of handling partial failure cases, this specification does not allow the inclusion of multiple mitigation requests in the same PUT request. Concretely, a DOTS client **MUST NOT** include multiple entries in the 'scope' array of the same PUT request.

FQDN and URI mitigation scopes may be thought of as a form of scope alias, in which the addresses associated with the domain name or URI (as resolved by the DOTS server) represent the scope of the mitigation. Particularly, the IP addresses to which the host subcomponent of authority component of a URI resolves represent the 'target-prefix', the URI scheme represents

the 'target-protocol', and the port subcomponent of authority component of a URI represents the 'target-port-range'. If the optional port information is not present in the authority component, the default port defined for the URI scheme represents the 'target-port'.

In the PUT request, at least one of the attributes 'target-prefix', 'target-fqdn','target-uri', or 'alias-name' **MUST** be present.

Attributes and Uri-Path parameters with empty values **MUST NOT** be present in a request, as an empty value will render the entire request invalid.

Figure 7 shows a PUT request example to signal that servers 2001:db8:6401::1 and 2001:db8:6401::2 are receiving attack traffic on TCP port numbers 80, 8080, and 443. The presence of 'cdid' indicates that a server-domain DOTS gateway has modified the initial PUT request sent by the DOTS client. Note that 'cdid' **MUST NOT** appear in the PUT request message body.

```
Header: PUT (Code=0.03)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "mitigate"
Uri-Path: "cdid=7eeaf349529eb55ed50113"
Uri-Path: "cuid=dz6pHjaADkaFTbjr0JGBpw"
Uri-Path: "mid=123"
Content-Format: "application/dots+cbor"

{
  "ietf-dots-signal-channel:mitigation-scope": {
    "scope": [
      {
        "target-prefix": [
           "2001:db8:6401::1/128",
           "2001:db8:6401::2/128"
        ],
        "target-port-range": [
          {
            "lower-port": 80
          },
          {
            "lower-port": 443
          },
          {
             "lower-port": 8080
          }
        ],
        "target-protocol": [
          6
        ],
        "lifetime": 3600
      }
    ]
  }
}
```

Figure 7: PUT for DOTS Mitigation Request (An Example)

The corresponding CBOR encoding format for the payload is shown in Figure 8.

```
A1                                      # map(1)
   01                                   # unsigned(1)
   A1                                   # map(1)
      02                                # unsigned(2)
      81                                # array(1)
         A4                             # map(4)
            06                          # unsigned(6)
            82                          # array(2)
               74                       # text(20)
                  323030313A6462383A363430313A3A312F313238
               74                       # text(20)
                  323030313A6462383A363430313A3A322F313238
            07                          # unsigned(7)
            83                          # array(3)
               A1                       # map(1)
                  08                    # unsigned(8)
                  18 50                 # unsigned(80)
               A1                       # map(1)
                  08                    # unsigned(8)
                  19 01BB               # unsigned(443)
               A1                       # map(1)
                  08                    # unsigned(8)
                  19 1F90               # unsigned(8080)
            0A                          # unsigned(10)
            81                          # array(1)
               06                       # unsigned(6)
            0E                          # unsigned(14)
            19 0E10                     # unsigned(3600)
```

Figure 8: PUT for DOTS Mitigation Request (CBOR)

4.4.1.2. Server-Domain DOTS Gateways

In deployments where server-domain DOTS gateways are enabled, identity information about the origin source client domain ('cdid') **SHOULD** be propagated to the DOTS server. That information is meant to assist the DOTS server in enforcing some policies, such as grouping DOTS clients that belong to the same DOTS domain, limiting the number of DOTS requests, and identifying the mitigation scope. These policies can be enforced per client, per client domain, or both. Also, the identity information may be used for auditing and debugging purposes.

Figure 9 shows an example of a request relayed by a server-domain DOTS gateway.

```
Header: PUT (Code=0.03)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "mitigate"
Uri-Path: "cdid=7eeaf349529eb55ed50113"
Uri-Path: "cuid=dz6pHjaADkaFTbjr0JGBpw"
Uri-Path: "mid=123"
Content-Format: "application/dots+cbor"

{
  ...
}
```

Figure 9: PUT for DOTS Mitigation Request as Relayed by a DOTS Gateway

A server-domain DOTS gateway **SHOULD** add the following Uri-Path parameter:

cdid: Stands for Client Domain Identifier. The 'cdid' is conveyed by a server-domain DOTS gateway to propagate the source domain identity from the gateway's client-facing side to the gateway's server-facing side and from the gateway's server-facing side to the DOTS server. 'cdid' may be used by the final DOTS server for policy-enforcement purposes (e.g., enforce a quota on filtering rules). These policies are deployment specific.

Server-domain DOTS gateways **SHOULD** support a configuration option to instruct whether the 'cdid' parameter is to be inserted.

In order to accommodate deployments that require enforcing per-client policies, per-client domain policies, or a combination thereof, server-domain DOTS gateways instructed to insert the 'cdid' parameter **MUST** supply the SPKI hash of the DOTS client X.509 certificate, the DOTS client raw public key, or the hash of the "PSK identity" in the 'cdid', following the same rules for generating the hash conveyed in 'cuid', which is then used by the ultimate DOTS server to determine the corresponding client's domain. The 'cdid' generated by a server-domain gateway is likely to be the same as the 'cuid' except the case in which the DOTS message was relayed by a client-domain DOTS gateway or the 'cuid' was generated by a rogue DOTS client.

If a DOTS client is provisioned, for example, with distinct certificates to use to peer with distinct server-domain DOTS gateways that peer to the same DOTS server, distinct 'cdid' values may be supplied by the server-domain DOTS gateways to the server. The ultimate DOTS server **MUST** treat those 'cdid' values as equivalent.

The 'cdid' attribute **MUST NOT** be generated and included by DOTS clients.

DOTS servers **MUST** ignore 'cdid' attributes that are directly supplied by source DOTS clients or client-domain DOTS gateways. This implies that first server-domain DOTS gateways **MUST** strip 'cdid' attributes supplied by DOTS clients. DOTS servers **SHOULD** support a configuration parameter to identify DOTS gateways that are trusted to supply 'cdid' attributes.

Only single-valued 'cdid' are defined in this document. That is, only the first on-path server-domain DOTS gateway can insert a 'cdid' value. This specification does not allow multiple server-domain DOTS gateways, whenever involved in the path, to insert a 'cdid' value for each server-domain gateway.

This is an optional Uri-Path. When present, 'cdid' **MUST** be positioned before 'cuid'.

A DOTS gateway **SHOULD** add the CoAP Hop-Limit Option [RFC8768].

4.4.1.3. Processing Mitigation Requests

The DOTS server couples the DOTS signal and data channel sessions using the DOTS client identity and optionally the 'cdid' parameter value, so the DOTS server can validate whether the aliases conveyed in the mitigation request were indeed created by the same DOTS client using the DOTS data channel session. If the aliases were not created by the DOTS client, the DOTS server **MUST** return 4.00 (Bad Request) in the response.

The DOTS server couples the DOTS signal channel sessions using the DOTS client identity and optionally the 'cdid' parameter value, and the DOTS server uses 'mid' and 'cuid' Uri-Path parameter values to detect duplicate mitigation requests. If the mitigation request contains the 'alias-name' and other parameters identifying the target resources (such as 'target-prefix', 'target-port-range', 'target-fqdn', or 'target-uri'), the DOTS server appends the parameter values associated with the 'alias-name' with the corresponding parameter values in 'target-prefix', 'target-port-range', 'target-fqdn', or 'target-uri'.

The DOTS server indicates the result of processing the PUT request using CoAP Response Codes. CoAP 2.xx codes are success. CoAP 4.xx codes are some sort of invalid requests (client errors). CoAP 5.xx codes are returned if the DOTS server is in an error state or is currently unavailable to provide mitigation in response to the mitigation request from the DOTS client.

Figure 10 shows an example response to a PUT request that is successfully processed by a DOTS server (i.e., CoAP 2.xx Response Codes). This version of the specification forbids 'cuid' and 'cdid' (if used) to be returned in a response message body.

```
{
  "ietf-dots-signal-channel:mitigation-scope": {
     "scope": [
        {
          "mid": 123,
          "lifetime": 3600
        }
      ]
   }
}
```

Figure 10: 2.xx Response Body

If the request is missing a mandatory attribute, does not include 'cuid' or 'mid' Uri-Path options, includes multiple 'scope' parameters, or contains invalid or unknown parameters, the DOTS server **MUST** reply with 4.00 (Bad Request). DOTS agents can safely ignore comprehension-optional parameters they don't understand (Section 10.6.1.1).

A DOTS server that receives a mitigation request with a 'lifetime' set to '0' **MUST** reply with a 4.00 (Bad Request).

If the DOTS server does not find the 'mid' parameter value conveyed in the PUT request in its configuration data, it **MAY** accept the mitigation request by sending back a 2.01 (Created) response to the DOTS client; the DOTS server will consequently try to mitigate the attack. A DOTS server **MAY** reject mitigation requests when it is near capacity or needs to rate-limit a particular client, for example.

The relative order of two mitigation requests with the same 'trigger- mitigation' type from a DOTS client is determined by comparing their respective 'mid' values. If two mitigation requests with the same 'trigger-mitigation' type have overlapping mitigation scopes, the mitigation request with the highest numeric 'mid' value will override the other mitigation request. Two mitigation requests from a DOTS client have overlapping scopes if there is a common IP address, IP prefix, FQDN, URI, or alias. To avoid maintaining a long list of overlapping mitigation requests (i.e., requests with the same 'trigger-mitigation' type and overlapping scopes) from a DOTS client and to avoid error-prone provisioning of mitigation requests from a DOTS client, the overlapped lower numeric 'mid' **MUST** be automatically deleted and no longer available at the DOTS server. For example, if the DOTS server receives a mitigation request that overlaps with an existing mitigation with a higher numeric 'mid', the DOTS server rejects the request by returning 4.09 (Conflict) to the DOTS client. The response **MUST** include enough information for a DOTS client to recognize the source of the conflict, as described below in the 'conflict-information' subtree (Section 5.1), with only the relevant nodes listed:

conflict-information: Indicates that a mitigation request is conflicting with another mitigation request. This attribute has the following structure:

conflict-cause: Indicates the cause of the conflict. The following value **MUST** be returned:

1: Overlapping targets. 'conflict-scope' provides more details about the conflicting target clauses.

conflict-scope: Characterizes the exact conflict scope. It may include a list of IP addresses, a list of prefixes, a list of target protocols, a list of FQDNs, a list of URIs, a list of aliases, or a 'mid'. A list of port numbers may also be included if there is a common IP address, IP prefix, FQDN, URI, or alias.

If the DOTS server receives a mitigation request that overlaps with an active mitigation request, but both have distinct 'trigger- mitigation' types, the DOTS server **SHOULD** deactivate (absent explicit policy/configuration otherwise) the mitigation request with 'trigger- mitigation' set to

'false'. Particularly, if the mitigation request with 'trigger-mitigation' set to 'false' is active, the DOTS server withdraws the mitigation request (i.e., status code is set to '7' as defined in Table 3) and transitions the status of the mitigation request to '8'.

Upon DOTS signal channel session loss with a peer DOTS client, the DOTS server **SHOULD** withdraw (absent explicit policy/configuration otherwise) any active mitigation requests that overlap with mitigation requests having 'trigger-mitigation' set to 'false' from that DOTS client, as the loss of the session implicitly activates these preconfigured mitigation requests, and they take precedence. Note that the active-but-terminating period is not observed for mitigations withdrawn at the initiative of the DOTS server.

DOTS clients may adopt various strategies for setting the scopes of immediate and preconfigured mitigation requests to avoid potential conflicts. For example, a DOTS client may tweak preconfigured scopes so that the scope of any overlapping immediate mitigation request will be a subset of the preconfigured scopes. Also, if an immediate mitigation request overlaps with any of the preconfigured scopes, the DOTS client sets the scope of the overlapping immediate mitigation request to be a subset of the preconfigured scopes, so as to get a broad mitigation when the DOTS signal channel collapses and to maximize the chance of recovery.

If the request conflicts with an existing mitigation request from a different DOTS client, the DOTS server may return 2.01 (Created) or 4.09 (Conflict) to the requesting DOTS client. If the DOTS server decides to maintain the new mitigation request, the DOTS server returns 2.01 (Created) to the requesting DOTS client. If the DOTS server decides to reject the new mitigation request, the DOTS server returns 4.09 (Conflict) to the requesting DOTS client. For both 2.01 (Created) and 4.09 (Conflict) responses, the response **MUST** include enough information for a DOTS client to recognize the source of the conflict as described below:

conflict-information: Indicates that a mitigation request is conflicting with another mitigation request(s) from other DOTS client(s). This attribute has the following structure:

conflict-status: Indicates the status of a conflicting mitigation request. The following values are defined:

1: DOTS server has detected conflicting mitigation requests from different DOTS clients. This mitigation request is currently inactive until the conflicts are resolved. Another mitigation request is active.

2: DOTS server has detected conflicting mitigation requests from different DOTS clients. This mitigation request is currently active.

3: DOTS server has detected conflicting mitigation requests from different DOTS clients. All conflicting mitigation requests are inactive.

conflict-cause: Indicates the cause of the conflict. The following values are defined:

1: Overlapping targets. 'conflict-scope' provides more details about the conflicting target clauses.

2: Conflicts with an existing accept-list. This code is returned when the DDoS mitigation detects source addresses/prefixes in the accept-listed Access Control Lists (ACLs) are attacking the target.

3: CUID Collision. This code is returned when a DOTS client uses a 'cuid' that is already used by another DOTS client. This code is an indication that the request has been rejected and a new request with a new 'cuid' is to be re-sent by the DOTS client (see the example shown in Figure 11). Note that 'conflict-status', 'conflict-scope', and 'retry-timer' **MUST NOT** be returned in the error response.

conflict-scope: Characterizes the exact conflict scope. It may include a list of IP addresses, a list of prefixes, a list of target protocols, a list of FQDNs, a list of URIs, a list of aliases, or references to conflicting ACLs (by an 'acl-name', typically [RFC8783]). A list of port numbers may also be included if there is a common IP address, IP prefix, FQDN, URI, or alias.

retry-timer: Indicates, in seconds, the time after which the DOTS client may reissue the same request. The DOTS server returns 'retry-timer' only to DOTS client(s) for which a mitigation request is deactivated. Any retransmission of the same mitigation request before the expiry of this timer is likely to be rejected by the DOTS server for the same reasons.

The 'retry-timer' **SHOULD** be equal to the lifetime of the active mitigation request resulting in the deactivation of the conflicting mitigation request.

If the DOTS server decides to maintain a state for the deactivated mitigation request, the DOTS server updates the lifetime of the deactivated mitigation request to 'retry-timer + 45 seconds' (that is, this mitigation request remains deactivated for the entire duration of 'retry-timer + 45 seconds') so that the DOTS client can refresh the deactivated mitigation request after 'retry-timer' seconds, but before the expiry of the lifetime, and check if the conflict is resolved.

```
(1) Request with a conflicting 'cuid'

Header: PUT (Code=0.03)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "mitigate"
Uri-Path: "cuid=dz6pHjaADkaFTbjr0JGBpw"
Uri-Path: "mid=12"

(2) Message body of the 4.09 (Conflict) response
    from the DOTS server

{
  "ietf-dots-signal-channel:mitigation-scope": {
     "scope": [
        {
          "conflict-information": {
            "conflict-cause": "cuid-collision"
           }
        }
      ]
   }
}

(3) Request with a new 'cuid'

Header: PUT (Code=0.03)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "mitigate"
Uri-Path: "cuid=f30d281ce6b64fc5a0b91e"
Uri-Path: "mid=12"
```

Figure 11: Example of Generating a New 'cuid'

As an active attack evolves, DOTS clients can adjust the scope of requested mitigation as necessary, by refining the scope of resources requiring mitigation. This can be achieved by sending a PUT request with a new 'mid' value that will override the existing one with overlapping mitigation scopes.

For a mitigation request to continue beyond the initial negotiated lifetime, the DOTS client has to refresh the current mitigation request by sending a new PUT request. This PUT request **MUST** use the same 'mid' value, and it **MUST** repeat all the other parameters as sent in the original mitigation request apart from a possible change to the 'lifetime' parameter value. In such a case, the DOTS server **MAY** update the mitigation request by setting the remaining lifetime to the newly negotiated lifetime, and a 2.04 (Changed) response is returned to indicate a successful update of the mitigation request. If this is not the case, the DOTS server **MUST** reject the request with a 4.00 (Bad Request).

4.4.2. Retrieve Information Related to a Mitigation

A GET request is used by a DOTS client to retrieve information (including status) of DOTS mitigations from a DOTS server.

'cuid' is a mandatory Uri-Path parameter for GET requests.

Uri-Path parameters with empty values **MUST NOT** be present in a request.

The same considerations for manipulating the 'cdid' parameter by server-domain DOTS gateways specified in Section 4.4.1 **MUST** be followed for GET requests.

The 'c' Uri-Query option is used to control selection of configuration and non-configuration data nodes. Concretely, the 'c' (content) parameter and its permitted values defined in Table 2 of [CORE-COMI] can be used to retrieve non-configuration data (attack mitigation status), configuration data, or both. The DOTS server **MAY** support this optional filtering capability. It can safely ignore it if not supported. If the DOTS client supports the optional filtering capability, it **SHOULD** use "c=n" query (to get back only the dynamically changing data) or "c=c" query (to get back the static configuration values) when the DDoS attack is active to limit the size of the response.

Value	Description
c	Return only configuration descendant data nodes
n	Return only non-configuration descendant data nodes
a	Return all descendant data nodes

Table 2: Permitted Values of the 'c' Parameter

The DOTS client can use block-wise transfer [RFC7959] to get the list of all its mitigations maintained by a DOTS server; it can send a Block2 Option in a GET request with NUM = 0 to aid in limiting the size of the response. If the representation of all the active mitigation requests associated with the DOTS client does not fit within a single datagram, the DOTS server **MUST** use the Block2 Option with NUM = 0 in the GET response. The Size2 Option may be conveyed in the response to indicate the total size of the resource representation. The DOTS client retrieves the rest of the representation by sending additional GET requests with Block2 Options containing NUM values greater than zero. The DOTS client **MUST** adhere to the block size preferences indicated by the DOTS server in the response. If the DOTS server uses the Block2 Option in the GET response, and the response is for a dynamically changing resource (e.g., "c=n" or "c=a" query), the DOTS server **MUST** include the ETag Option in the response. The DOTS client **MUST** include the same ETag value in subsequent GET requests to retrieve the rest of the representation.

The following examples illustrate how a DOTS client retrieves active mitigation requests from a DOTS server. In particular:

- Figure 12 shows the example of a GET request to retrieve all DOTS mitigation requests signaled by a DOTS client.
- Figure 13 shows the example of a GET request to retrieve a specific DOTS mitigation request signaled by a DOTS client. The configuration data to be reported in the response is formatted in the same order as it was processed by the DOTS server in the original mitigation request.

These two examples assume the default of "c=a"; that is, the DOTS client asks for all data to be reported by the DOTS server.

```
Header: GET (Code=0.01)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "mitigate"
Uri-Path: "cuid=dz6pHjaADkaFTbjr0JGBpw"
Observe: 0
```

Figure 12: GET to Retrieve All DOTS Mitigation Requests

```
Header: GET (Code=0.01)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "mitigate"
Uri-Path: "cuid=dz6pHjaADkaFTbjr0JGBpw"
Uri-Path: "mid=12332"
Observe: 0
```

Figure 13: GET to Retrieve a Specific DOTS Mitigation Request

If the DOTS server does not find the 'mid' Uri-Path value conveyed in the GET request in its configuration data for the requesting DOTS client, it **MUST** respond with a 4.04 (Not Found) error Response Code. Likewise, the same error **MUST** be returned as a response to a request to retrieve all mitigation records (i.e., 'mid' Uri-Path is not defined) of a given DOTS client if the DOTS server does not find any mitigation record for that DOTS client. As a reminder, a DOTS client is identified by its identity (e.g., client certificate, 'cuid') and optionally the 'cdid'.

Figure 14 shows a response example of all active mitigation requests associated with the DOTS client, as maintained by the DOTS server. The response indicates the mitigation status of each mitigation request.

```
{
  "ietf-dots-signal-channel:mitigation-scope": {
    "scope": [
      {
        "mid": 12332,
        "mitigation-start": "1507818434",
        "target-prefix": [
             "2001:db8:6401::1/128",
             "2001:db8:6401::2/128"
        ],
        "target-protocol": [
          17
        ],
        "lifetime": 1756,
        "status": "attack-successfully-mitigated",
        "bytes-dropped": "134334555",
        "bps-dropped": "43344",
        "pkts-dropped": "333334444",
        "pps-dropped": "432432"
      },
      {
        "mid": 12333,
        "mitigation-start": "1507818393",
        "target-prefix": [
             "2001:db8:6401::1/128",
             "2001:db8:6401::2/128"
        ],
        "target-protocol": [
          6
        ],
        "lifetime": 1755,
        "status": "attack-stopped",
        "bytes-dropped": "0",
        "bps-dropped": "0",
        "pkts-dropped": "0",
        "pps-dropped": "0"
      }
    ]
  }
}
```

Figure 14: Response Body to a GET Request

The mitigation status parameters are described below:

mitigation-start: Mitigation start time is expressed in seconds relative to 1970-01-01T00:00Z in UTC time (Section 3.4.1 of [RFC8949]). The CBOR encoding is modified so that the leading tag 1 (epoch-based date/time) **MUST** be omitted.

This is a mandatory attribute when an attack mitigation is active. Particularly, 'mitigation-start' is not returned for a mitigation with 'status' code set to 8.

lifetime: The remaining lifetime of the mitigation request, in seconds.

This is a mandatory attribute.

status: Status of attack mitigation. The various possible values of 'status' parameter are explained in Table 3.

This is a mandatory attribute.

bytes-dropped: The total dropped byte count for the mitigation request since the attack mitigation was triggered. The count wraps around when it reaches the maximum value of unsigned integer64.

This is an optional attribute.

bps-dropped: The average number of dropped bytes per second for the mitigation request since the attack mitigation was triggered. This average **SHOULD** be over five-minute intervals (that is, measuring bytes into five-minute buckets and then averaging these buckets over the time since the mitigation was triggered).

This is an optional attribute.

pkts-dropped: The total number of dropped packet count for the mitigation request since the attack mitigation was triggered. The count wraps around when it reaches the maximum value of unsigned integer64.

This is an optional attribute.

pps-dropped: The average number of dropped packets per second for the mitigation request since the attack mitigation was triggered. This average **SHOULD** be over five-minute intervals (that is, measuring packets into five-minute buckets and then averaging these buckets over the time since the mitigation was triggered).

This is an optional attribute.

Parameter Value	Description
1	Attack mitigation setup is in progress (e.g., changing the network path to redirect the inbound traffic to a DOTS mitigator).
2	Attack is being successfully mitigated (e.g., traffic is redirected to a DDoS mitigator and attack traffic is dropped).
3	Attack has stopped and the DOTS client can withdraw the mitigation request. This status code will be transmitted for immediate mitigation requests till the mitigation is withdrawn or the lifetime expires. For mitigation requests with preconfigured scopes (i.e., 'trigger-mitigation' set to 'false'), this status code will be transmitted four times and then transition to '8'.

Parameter Value	Description
4	Attack has exceeded the mitigation provider capability.
5	DOTS client has withdrawn the mitigation request and the mitigation is active but terminating.
6	Attack mitigation is now terminated.
7	Attack mitigation is withdrawn (by the DOTS server). If a mitigation request with 'trigger- mitigation' set to 'false' is withdrawn because it overlaps with an immediate mitigation request, this status code will be transmitted four times and then transition to '8' for the mitigation request with preconfigured scopes.
8	Attack mitigation will be triggered for the mitigation request only when the DOTS signal channel session is lost.

Table 3: Values of 'status' Parameter

4.4.2.1. DOTS Servers Sending Mitigation Status

The Observe Option defined in [RFC7641] extends the CoAP core protocol with a mechanism for a CoAP client to "observe" a resource on a CoAP server: the client retrieves a representation of the resource and requests this representation be updated by the server as long as the client is interested in the resource. DOTS implementations **MUST** support the Observe Option for both 'mitigate' and 'config' (Section 4.2).

A DOTS client conveys the Observe Option set to '0' in the GET request to receive asynchronous notifications of attack mitigation status from the DOTS server.

Unidirectional mitigation notifications within the bidirectional signal channel enables asynchronous notifications between the agents. [RFC7641] indicates that (1) a notification can be sent in a Confirmable or a Non-confirmable message and (2) the message type used is typically application dependent and may be determined by the server for each notification individually. For the DOTS server application, the message type **MUST** always be set to Non-confirmable even if the underlying CoAP library elects a notification to be sent in a Confirmable message. This overrides the behavior defined in Section 4.5 of [RFC7641] to send a Confirmable message instead of a Non-confirmable message at least every 24 hours for the following reasons: First, the DOTS signal channel uses a heartbeat mechanism to determine if the DOTS client is alive. Second, Confirmable messages are not suitable during an attack.

Due to the higher likelihood of packet loss during a DDoS attack, the DOTS server periodically sends attack mitigation status to the DOTS client and also notifies the DOTS client whenever the status of the attack mitigation changes. If the DOTS server cannot maintain an RTT estimate, it **MUST NOT** send more than one asynchronous notification every 3 seconds and **SHOULD** use an even less aggressive rate whenever possible (case 2 in Section 3.1.3 of [RFC8085]).

When conflicting requests are detected, the DOTS server enforces the corresponding policy (e.g., accept all requests, reject all requests, accept only one request but reject all the others). It is assumed that this policy is supplied by the DOTS server administrator or that it is a default behavior of the DOTS server implementation. Then, the DOTS server sends a notification message (s) to the DOTS client(s) at the origin of the conflict (refer to the conflict parameters defined in Section 4.4.1). A conflict notification message includes information about the conflict cause, scope, and the status of the mitigation request(s). For example:

- A notification message with 'status' code set to '7 (Attack mitigation is withdrawn)' and 'conflict-status' set to '1' is sent to a DOTS client to indicate that an active mitigation request is deactivated because a conflict is detected.
- A notification message with 'status' code set to '1 (Attack mitigation is in progress)' and 'conflict-status' set to '2' is sent to a DOTS client to indicate that this mitigation request is in progress, but a conflict is detected.

Upon receipt of a conflict notification message indicating that a mitigation request is deactivated because of a conflict, a DOTS client **MUST NOT** resend the same mitigation request before the expiry of 'retry-timer'. It is also recommended that DOTS clients support the means to alert administrators about mitigation conflicts.

A DOTS client that is no longer interested in receiving notifications from the DOTS server can simply "forget" the observation. When the DOTS server sends the next notification, the DOTS client will not recognize the token in the message and, thus, will return a Reset message. This causes the DOTS server to remove the associated entry. Alternatively, the DOTS client can explicitly de-register itself by issuing a GET request that has the Token field set to the token of the observation to be canceled and includes an Observe Option with the value set to '1' (de-register). The latter is more deterministic and, thus, is **RECOMMENDED**.

Figure 15 shows an example of a DOTS client requesting a DOTS server to send notifications related to a mitigation request. Note that for mitigations with preconfigured scopes (i.e., 'trigger-mitigation' set to 'false'), the state will need to transition from '3' (attack-stopped) to '8' (attack-mitigation-signal-loss).

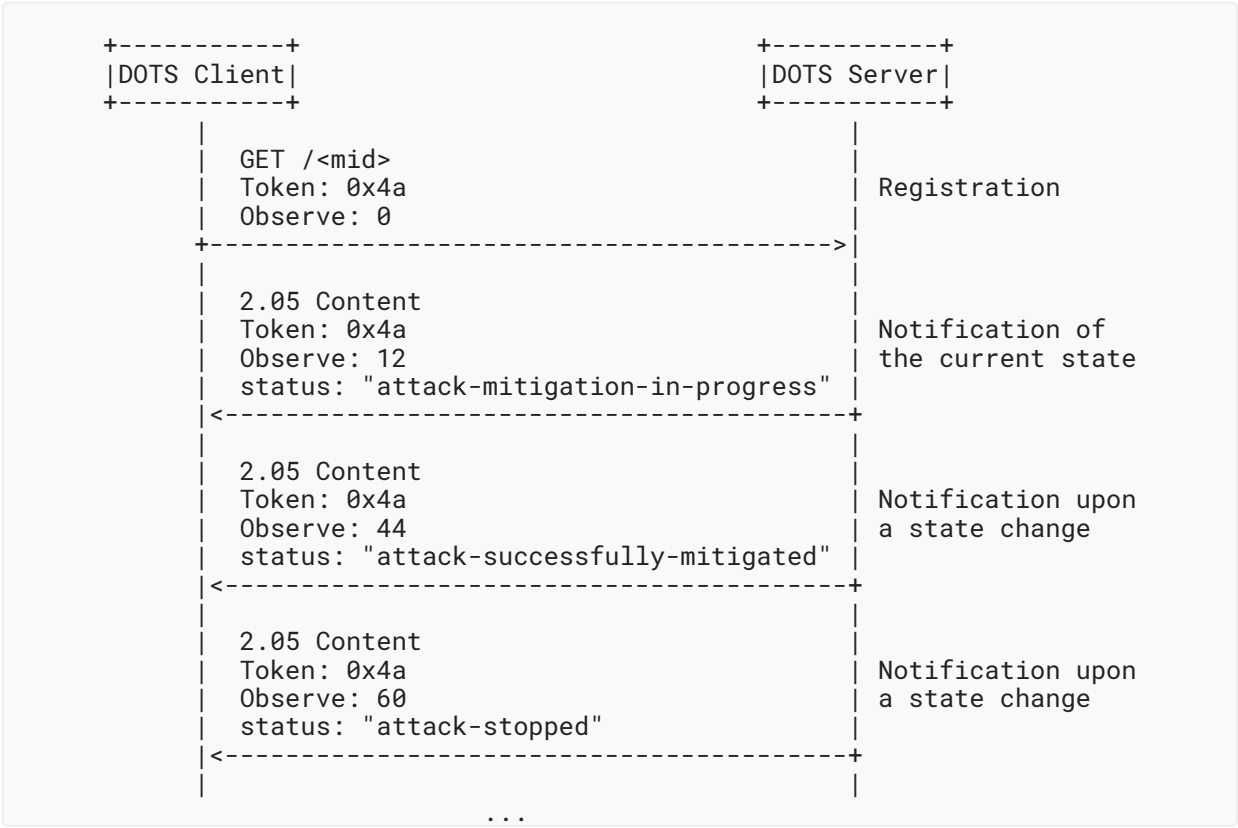

Figure 15: Notifications of Attack Mitigation Status

4.4.2.2. DOTS Clients Polling for Mitigation Status

The DOTS client can send the GET request at frequent intervals without the Observe Option to retrieve the configuration data of the mitigation request and non-configuration data (i.e., the attack status). DOTS clients **MAY** be configured with a policy indicating the frequency of polling DOTS servers to get the mitigation status. This frequency **MUST NOT** be more than one UDP datagram per RTT, as discussed in Section 3.1.3 of [RFC8085].

If the DOTS server has been able to mitigate the attack and the attack has stopped, the DOTS server indicates as such in the status. In such case, the DOTS client withdraws the mitigation request by issuing a DELETE request for this mitigation request (Section 4.4.4).

A DOTS client **SHOULD** react to the status of the attack per the information sent by the DOTS server rather than performing its own detection that the attack has been mitigated. This ensures that the DOTS client does not withdraw a mitigation request prematurely because it is possible that the DOTS client does not sense the DDoS attack on its resources, but the DOTS server could be actively mitigating the attack because the attack is not completely averted.

4.4.3. Efficacy Update from DOTS Clients

While DDoS mitigation is in progress, due to the likelihood of packet loss, a DOTS client **MAY** periodically transmit DOTS mitigation efficacy updates to the relevant DOTS server. A PUT request is used to convey the mitigation efficacy update to the DOTS server. This PUT request is treated as a refresh of the current mitigation.

The 'attack-status' parameter is a mandatory attribute when performing an efficacy update. The various possible values contained in the 'attack-status' parameter are described in Table 4.

Parameter Value	Description
1	The DOTS client determines that it is still under attack.
2	The DOTS client determines that the attack is successfully mitigated (e.g., attack traffic is not seen).

Table 4: Values of 'attack-status' Parameter

The PUT request used for the efficacy update **MUST** include all the parameters used in the PUT request to carry the DOTS mitigation request (Section 4.4.1) unchanged apart from the 'lifetime' parameter value. If this is not the case, the DOTS server **MUST** reject the request with a 4.00 (Bad Request).

The If-Match Option (Section 5.10.8.1 of [RFC7252]) with an empty value is used to make the PUT request conditional on the current existence of the mitigation request. If UDP is used as transport, CoAP requests may arrive out of order. For example, the DOTS client may send a PUT request to convey an efficacy update to the DOTS server followed by a DELETE request to withdraw the mitigation request, but the DELETE request arrives at the DOTS server before the PUT request. To handle out-of-order delivery of requests, if an If-Match Option is present in the PUT request and the 'mid' in the request matches a mitigation request from that DOTS client, the request is processed by the DOTS server. If no match is found, the PUT request is silently ignored by the DOTS server.

An example of an efficacy update message, which includes an If-Match Option with an empty value, is depicted in Figure 16.

```
Header: PUT (Code=0.03)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "mitigate"
Uri-Path: "cuid=dz6pHjaADkaFTbjr0JGBpw"
Uri-Path: "mid=123"
If-Match:
Content-Format: "application/dots+cbor"

{
 "ietf-dots-signal-channel:mitigation-scope": {
   "scope": [
     {
       "target-prefix": [
          "2001:db8:6401::1/128",
          "2001:db8:6401::2/128"
        ],
       "target-port-range": [
         {
           "lower-port": 80
         },
         {
           "lower-port": 443
         },
         {
            "lower-port": 8080
         }
       ],
       "target-protocol": [
          6
       ],
       "attack-status": "under-attack"
     }
   ]
 }
}
```

Figure 16: An Example of Efficacy Update

The DOTS server indicates the result of processing a PUT request using CoAP Response Codes. The Response Code 2.04 (Changed) is returned if the DOTS server has accepted the mitigation efficacy update. The error Response Code 5.03 (Service Unavailable) is returned if the DOTS server has erred or is incapable of performing the mitigation. As specified in [RFC7252], 5.03 uses Max-Age Option to indicate the number of seconds after which to retry.

4.4.4. Withdraw a Mitigation

DELETE requests are used to withdraw DOTS mitigation requests from DOTS servers (Figure 17).

'cuid' and 'mid' are mandatory Uri-Path parameters for DELETE requests.

The same considerations for manipulating the 'cdid' parameter by DOTS gateways, as specified in Section 4.4.1, **MUST** be followed for DELETE requests. Uri-Path parameters with empty values **MUST NOT** be present in a request.

```
Header: DELETE (Code=0.04)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "mitigate"
Uri-Path: "cuid=dz6pHjaADkaFTbjr0JGBpw"
Uri-Path: "mid=123"
```

Figure 17: Withdraw a DOTS Mitigation

If the DELETE request does not include 'cuid' and 'mid' parameters, the DOTS server **MUST** reply with a 4.00 (Bad Request).

Once the request is validated, the DOTS server immediately acknowledges a DOTS client's request to withdraw the DOTS mitigation request using a 2.02 (Deleted) Response Code with no response payload. A 2.02 (Deleted) Response Code is returned even if the 'mid' parameter value conveyed in the DELETE request does not exist in its configuration data before the request.

If the DOTS server finds the 'mid' parameter value conveyed in the DELETE request in its configuration data for the DOTS client, then to protect against route or DNS flapping caused by a DOTS client rapidly removing a mitigation and to dampen the effect of oscillating attacks, the DOTS server **MAY** allow mitigation to continue for a limited period after acknowledging a DOTS client's withdrawal of a mitigation request. During this period, the DOTS server status messages **SHOULD** indicate that mitigation is active but terminating (Section 4.4.2).

The initial active-but-terminating period **SHOULD** be sufficiently long to absorb latency incurred by route propagation. The active-but-terminating period **SHOULD** be set by default to 120 seconds. If the client requests mitigation again before the initial active-but-terminating period elapses, the DOTS server **MAY** exponentially increase (the base of the exponent is 2) the active-but-terminating period up to a maximum of 300 seconds (5 minutes).

Once the active-but-terminating period elapses, the DOTS server **MUST** treat the mitigation as terminated.

If a mitigation is triggered due to a signal channel loss, the DOTS server relies upon normal triggers to stop that mitigation (typically, receipt of a valid DELETE request, expiry of the mitigation lifetime, or scrubbing the traffic to the attack target). In particular, the DOTS server **MUST NOT** consider the signal channel recovery as a trigger to stop the mitigation.

4.5. DOTS Signal Channel Session Configuration

A DOTS client can negotiate, configure, and retrieve the DOTS signal channel session behavior with its DOTS peers. The DOTS signal channel can be used, for example, to configure the following:

a. Heartbeat interval ('heartbeat-interval'): DOTS agents regularly send heartbeats to each other after mutual authentication is successfully completed in order to keep the DOTS signal channel open. Heartbeat messages are exchanged between DOTS agents every 'heartbeat-interval' seconds to detect the current status of the DOTS signal channel session.

b. Missing heartbeats allowed ('missing-hb-allowed'): This variable indicates the maximum number of consecutive heartbeat messages for which a DOTS agent did not receive a response before concluding that the session is disconnected or defunct.

c. Acceptable probing rate ('probing-rate'): This parameter indicates the average data rate that must not be exceeded by a DOTS agent in sending to a peer DOTS agent that does not respond.

d. Acceptable signal loss ratio: Maximum retransmissions ('max-retransmit'), retransmission timeout value ('ack-timeout'), and other message transmission parameters for Confirmable messages over the DOTS signal channel.

When the DOTS signal channel is established over a reliable transport (e.g., TCP), there is no need for the reliability mechanisms provided by CoAP over UDP since the underlying TCP connection provides retransmissions and deduplication [RFC8323]. CoAP over reliable transports does not support Confirmable or Non-confirmable message types. As such, the transmission-related parameters ('missing-hb-allowed' and acceptable signal loss ratio) are negotiated only for DOTS over unreliable transports.

The same or distinct configuration sets may be used during times when a mitigation is active ('mitigating-config') and when no mitigation is active ('idle-config'). This is particularly useful for DOTS servers that might want to reduce heartbeat frequency or cease heartbeat exchanges when an active DOTS client has not requested mitigation. If distinct configurations are used, DOTS agents **MUST** follow the appropriate configuration set as a function of the mitigation activity (e.g., if no mitigation request is active (also referred to as 'idle' time), values related to 'idle-config' must be followed). Additionally, DOTS agents **MUST** automatically switch to the other configuration upon a change in the mitigation activity (e.g., if an attack mitigation is launched after an 'idle' time, the DOTS agent switches from values related to 'idle-config' to values related to 'mitigating-config').

CoAP requests and responses are indicated for reliable delivery by marking them as Confirmable messages. DOTS signal channel session configuration requests and responses are marked as Confirmable messages. As explained in Section 2.1 of [RFC7252], a Confirmable message is retransmitted using a default timeout and exponential backoff between retransmissions until the DOTS server sends an Acknowledgement message (ACK) with the same Message ID conveyed from the DOTS client.

Message transmission parameters are defined in Section 4.8 of [RFC7252]. The DOTS server can either piggyback the response in the Acknowledgement message or, if the DOTS server cannot respond immediately to a request carried in a Confirmable message, it simply responds with an Empty Acknowledgement message so that the DOTS client can stop retransmitting the request. Empty Acknowledgement messages are explained in Section 2.2 of [RFC7252]. When the response is ready, the server sends it in a new Confirmable message, which, in turn, needs to be acknowledged by the DOTS client (see Sections 5.2.1 and 5.2.2 of [RFC7252]). Requests and responses exchanged between DOTS agents during 'idle' time, except heartbeat messages, are marked as Confirmable messages.

> Implementation Note: A DOTS client that receives a response in a Confirmable message may want to clean up the message state right after sending the ACK. If that ACK is lost and the DOTS server retransmits the Confirmable message, the DOTS client may no longer have any state that would help it correlate this response; from the DOTS client's standpoint, the retransmission message is unexpected. The DOTS client will send a Reset message so it does not receive any more retransmissions. This behavior is normal and not an indication of an error (see Section 5.3.2 of [RFC7252] for more details).

4.5.1. Discover Configuration Parameters

A GET request is used to obtain acceptable (e.g., minimum and maximum values) and current configuration parameters on the DOTS server for DOTS signal channel session configuration. This procedure occurs between a DOTS client and its immediate peer DOTS server. As such, this GET request **MUST NOT** be relayed by a DOTS gateway.

Figure 18 shows how to obtain configuration parameters that the DOTS server will find acceptable.

```
Header: GET (Code=0.01)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "config"
```

Figure 18: GET to Retrieve Configuration

The DOTS server in the 2.05 (Content) response conveys the current, minimum, and maximum attribute values acceptable by the DOTS server (Figure 19).

```
{
  "ietf-dots-signal-channel:signal-config": {
    "mitigating-config": {
      "heartbeat-interval": {
        "max-value": number,
        "min-value": number,
        "current-value": number
      },
      "missing-hb-allowed": {
        "max-value": number,
        "min-value": number,
        "current-value": number
      },
      "probing-rate": {
        "max-value": number,
        "min-value": number,
        "current-value": number
      },
      "max-retransmit": {
        "max-value": number,
        "min-value": number,
        "current-value": number
      },
      "ack-timeout": {
        "max-value-decimal": "string",
        "min-value-decimal": "string",
        "current-value-decimal": "string"
      },
      "ack-random-factor": {
        "max-value-decimal": "string",
        "min-value-decimal": "string",
        "current-value-decimal": "string"
      }
    },
    "idle-config": {
      "heartbeat-interval": {
        "max-value": number,
        "min-value": number,
        "current-value": number
      },
      "missing-hb-allowed": {
        "max-value": number,
        "min-value": number,
        "current-value": number
      },
      "probing-rate": {
        "max-value": number,
        "min-value": number,
        "current-value": number
      },
      "max-retransmit": {
        "max-value": number,
        "min-value": number,
        "current-value": number
      },
      "ack-timeout": {
        "max-value-decimal": "string",
```

```
          "min-value-decimal": "string",
          "current-value-decimal": "string"
        },
        "ack-random-factor": {
          "max-value-decimal": "string",
          "min-value-decimal": "string",
          "current-value-decimal": "string"
        }
      }
    }
  }
```

Figure 19: GET Configuration Response Body Schema

The parameters in Figure 19 are described below:

mitigating-config: Set of configuration parameters to use when a mitigation is active. The following parameters may be included:

heartbeat-interval: Time interval in seconds between two consecutive heartbeat messages.

'0' is used to disable the heartbeat mechanism.

This is an optional attribute.

missing-hb-allowed: Maximum number of consecutive heartbeat messages for which the DOTS agent did not receive a response before concluding that the session is disconnected.

This is an optional attribute.

probing-rate: The average data rate, in bytes/second, that must not be exceeded by a DOTS agent in sending to a peer DOTS agent that does not respond (referred to as PROBING_RATE parameter in CoAP).

This is an optional attribute.

max-retransmit: Maximum number of retransmissions for a message (referred to as MAX_RETRANSMIT parameter in CoAP).

This is an optional attribute.

ack-timeout: Timeout value in seconds used to calculate the initial retransmission timeout value (referred to as ACK_TIMEOUT parameter in CoAP).

This is an optional attribute.

ack-random-factor: Random factor used to influence the timing of retransmissions (referred to as ACK_RANDOM_FACTOR parameter in CoAP).

This is an optional attribute.

idle-config: Set of configuration parameters to use when no mitigation is active. This attribute has the same structure as 'mitigating-config'.

Figure 20 shows an example of acceptable and current configuration parameters on a DOTS server for DOTS signal channel session configuration. The same acceptable configuration is used during mitigation and idle times.

```
{
  "ietf-dots-signal-channel:signal-config": {
    "mitigating-config": {
      "heartbeat-interval": {
        "max-value": 240,
        "min-value": 15,
        "current-value": 30
      },
      "missing-hb-allowed": {
        "max-value": 20,
        "min-value": 3,
        "current-value": 15
      },
      "probing-rate": {
        "max-value": 20,
        "min-value": 5,
        "current-value": 15
      },
      "max-retransmit": {
        "max-value": 15,
        "min-value": 2,
        "current-value": 3
      },
      "ack-timeout": {
        "max-value-decimal": "30.00",
        "min-value-decimal": "1.00",
        "current-value-decimal": "2.00"
      },
      "ack-random-factor": {
        "max-value-decimal": "4.00",
        "min-value-decimal": "1.10",
        "current-value-decimal": "1.50"
      }
    },
    "idle-config": {
      "heartbeat-interval": {
        "max-value": 240,
        "min-value": 15,
        "current-value": 30
      },
      "missing-hb-allowed": {
        "max-value": 20,
        "min-value": 3,
        "current-value": 15
      },
      "probing-rate": {
        "max-value": 20,
        "min-value": 5,
        "current-value": 15
      },
      "max-retransmit": {
        "max-value": 15,
        "min-value": 2,
        "current-value": 3
      },
      "ack-timeout": {
        "max-value-decimal": "30.00",
```

```
        "min-value-decimal": "1.00",
        "current-value-decimal": "2.00"
      },
      "ack-random-factor": {
        "max-value-decimal": "4.00",
        "min-value-decimal": "1.10",
        "current-value-decimal": "1.50"
      }
    }
  }
}
```

Figure 20: Example of a Configuration Response Body

4.5.2. Convey DOTS Signal Channel Session Configuration

A PUT request (Figures 21 and 22) is used to convey the configuration parameters for the signal channel (e.g., heartbeat interval, maximum retransmissions). Message transmission parameters for CoAP are defined in Section 4.8 of [RFC7252]. The **RECOMMENDED** values of transmission parameter values are 'ack-timeout' (2 seconds), 'max-retransmit' (3), and 'ack-random-factor' (1.5). In addition to those parameters, the **RECOMMENDED** specific DOTS transmission parameter values are 'heartbeat-interval' (30 seconds) and 'missing-hb-allowed' (15).

> Note: 'heartbeat-interval' should be tweaked to also assist DOTS messages for NAT traversal (SIG-011 of [RFC8612]). According to [RFC8085], heartbeat messages must not be sent more frequently than once every 15 seconds and should use longer intervals when possible. Furthermore, [RFC4787] recommends that NATs use a state timeout of 2 minutes or longer, but experience shows that sending packets every 15 to 30 seconds is necessary to prevent the majority of middleboxes from losing state for UDP flows. From that standpoint, the **RECOMMENDED** minimum 'heartbeat-interval' is 15 seconds and the **RECOMMENDED** maximum 'heartbeat-interval' is 240 seconds. The recommended value of 30 seconds is selected to anticipate the expiry of NAT state.
>
> A 'heartbeat-interval' of 30 seconds may be considered to be too chatty in some deployments. For such deployments, DOTS agents may negotiate longer 'heartbeat-interval' values to prevent any network overload with too frequent heartbeats.
>
> Different heartbeat intervals can be defined for 'mitigating- config' and 'idle-config' to reduce being too chatty during idle times. If there is an on-path translator between the DOTS client (standalone or part of a DOTS gateway) and the DOTS server, the 'mitigating-config' 'heartbeat-interval' has to be smaller than the translator session timeout. It is recommended that the 'idle-config' 'heartbeat-interval' also be smaller than the translator session timeout to prevent translator traversal issues or that it be disabled entirely. Means to discover the lifetime assigned by a translator are out of scope.
>
> Given that the size of the heartbeat request cannot exceed ('heartbeat-interval' * 'probing-rate') bytes, 'probing-rate' should be set appropriately to avoid slowing down heartbeat exchanges. For example, 'probing-rate' may be set to 2 * ("size of

> encrypted DOTS heartbeat request"/'heartbeat- interval') or (("size of encrypted DOTS heartbeat request" + "average size of an encrypted mitigation request")/'heartbeat- interval'). Absent any explicit configuration or inability to dynamically adjust 'probing-rate' values (Section 4.8.1 of [RFC7252]), DOTS agents use 5 bytes/second as a default 'probing-rate' value.

If the DOTS agent wishes to change the default values of message transmission parameters, it **SHOULD** follow the guidance given in Section 4.8.1 of [RFC7252]. The DOTS agents **MUST** use the negotiated values for message transmission parameters and default values for non-negotiated message transmission parameters.

The signal channel session configuration is applicable to a single DOTS signal channel session between DOTS agents, so the 'cuid' Uri-Path **MUST NOT** be used.

```
Header: PUT (Code=0.03)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "config"
Uri-Path: "sid=123"
Content-Format: "application/dots+cbor"

{
 ...
}
```

Figure 21: PUT to Convey the DOTS Signal Channel Session Configuration Data

The additional Uri-Path parameter to those defined in Table 1 is as follows:

sid: Session Identifier is an identifier for the DOTS signal channel session configuration data represented as an integer. This identifier **MUST** be generated by DOTS clients. 'sid' values **MUST** increase monotonically (when a new PUT is generated by a DOTS client to convey the configuration parameters for the signal channel).

This is a mandatory attribute.

```
{
  "ietf-dots-signal-channel:signal-config": {
    "mitigating-config": {
      "heartbeat-interval": {
        "current-value": number
      },
      "missing-hb-allowed": {
        "current-value": number
      },
      "probing-rate": {
        "current-value": number
      },
      "max-retransmit": {
        "current-value": number
      },
      "ack-timeout": {
        "current-value-decimal": "string"
      },
      "ack-random-factor": {
        "current-value-decimal": "string"
      }
    },
    "idle-config": {
      "heartbeat-interval": {
        "current-value": number
      },
      "missing-hb-allowed": {
        "current-value": number
      },
      "probing-rate": {
        "current-value": number
      },
      "max-retransmit": {
        "current-value": number
      },
      "ack-timeout": {
        "current-value-decimal": "string"
      },
      "ack-random-factor": {
        "current-value-decimal": "string"
      }
    }
  }
}
```

Figure 22: PUT to Convey the DOTS Signal Channel Session Configuration Data (Message Body Schema)

The meaning of the parameters in the CBOR body (Figure 22) is defined in Section 4.5.1.

At least one of the attributes 'heartbeat-interval', 'missing-hb-allowed', 'probing-rate', 'max-retransmit', 'ack-timeout', and 'ack-random-factor' **MUST** be present in the PUT request. Note that 'heartbeat-interval', 'missing-hb-allowed', 'probing-rate', 'max-retransmit', 'ack-timeout', and 'ack-

random-factor', if present, do not need to be provided for both 'mitigating-config' and 'idle-config' in a PUT request. A request does not need to include both 'mitigating-config' and 'idle-config' attributes.

The PUT request with a higher numeric 'sid' value overrides the DOTS signal channel session configuration data installed by a PUT request with a lower numeric 'sid' value. That is, the configuration parameters that are included in the PUT request with a higher numeric 'sid' value will be used instead of the DOTS server's defaults. To avoid maintaining a long list of 'sid' requests from a DOTS client, the lower numeric 'sid' **MUST** be automatically deleted and no longer available at the DOTS server.

Figure 23 shows a PUT request example to convey the configuration parameters for the DOTS signal channel. In this example, the heartbeat mechanism is disabled when no mitigation is active, while the heartbeat interval is set to '30' when a mitigation is active.

```
Header: PUT (Code=0.03)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "config"
Uri-Path: "sid=123"
Content-Format: "application/dots+cbor"

{
  "ietf-dots-signal-channel:signal-config": {
    "mitigating-config": {
      "heartbeat-interval": {
        "current-value": 30
      },
      "missing-hb-allowed": {
        "current-value": 15
      },
      "probing-rate": {
        "current-value": 15
      },
      "max-retransmit": {
        "current-value": 3
      },
      "ack-timeout": {
        "current-value-decimal": "2.00"
      },
      "ack-random-factor": {
        "current-value-decimal": "1.50"
      }
    },
    "idle-config": {
      "heartbeat-interval": {
        "current-value": 0
      },
      "max-retransmit": {
        "current-value": 3
      },
      "ack-timeout": {
        "current-value-decimal": "2.00"
      },
      "ack-random-factor": {
        "current-value-decimal": "1.50"
      }
    }
  }
}
```

Figure 23: PUT to Convey the Configuration Parameters

The DOTS server indicates the result of processing the PUT request using CoAP Response Codes:

- If the request is missing a mandatory attribute, does not include a 'sid' Uri-Path, or contains one or more invalid or unknown parameters, 4.00 (Bad Request) **MUST** be returned in the response.

- If the DOTS server does not find the 'sid' parameter value conveyed in the PUT request in its configuration data and if the DOTS server has accepted the configuration parameters, then a Response Code 2.01 (Created) **MUST** be returned in the response.
- If the DOTS server finds the 'sid' parameter value conveyed in the PUT request in its configuration data and if the DOTS server has accepted the updated configuration parameters, 2.04 (Changed) **MUST** be returned in the response.
- If any of the 'heartbeat-interval', 'missing-hb-allowed', 'probing-rate', 'max-retransmit', 'target-protocol', 'ack-timeout', and 'ack-random-factor' attribute values are not acceptable to the DOTS server, 4.22 (Unprocessable Entity) **MUST** be returned in the response. Upon receipt of this error code, the DOTS client **SHOULD** retrieve the maximum and minimum attribute values acceptable to the DOTS server (Section 4.5.1).

  The DOTS client may retry and send the PUT request with updated attribute values acceptable to the DOTS server.

A DOTS client may issue a GET message for 'config' with a 'sid' Uri-Path parameter to retrieve the negotiated configuration. The response does not need to include 'sid' in its message body.

4.5.3. Configuration Freshness and Notifications

Max-Age Option (Section 5.10.5 of [RFC7252]) **SHOULD** be returned by a DOTS server to associate a validity time with a configuration it sends. This feature forces the client to retrieve the updated configuration data if a change occurs at the DOTS server side. For example, the new configuration may instruct a DOTS client to cease heartbeats or reduce heartbeat frequency.

It is **NOT RECOMMENDED** to return a Max-Age Option set to 0.

Returning a Max-Age Option set to $2^{(32)}$-1 is equivalent to associating an infinite lifetime with the configuration.

If a non-zero value of Max-Age Option is received by a DOTS client, it **MUST** issue a GET request with a 'sid' Uri-Path parameter to retrieve the current and acceptable configuration before the expiry of the value enclosed in the Max-Age Option. This request is considered by the client and the server to be a means to refresh the configuration parameters for the signal channel. When a DDoS attack is active, refresh requests **MUST NOT** be sent by DOTS clients, and the DOTS server **MUST NOT** terminate the (D)TLS session after the expiry of the value returned in Max-Age Option.

If Max-Age Option is not returned in a response, the DOTS client initiates GET requests to refresh the configuration parameters each 60 seconds (Section 5.10.5 of [RFC7252]). To prevent such overload, it is **RECOMMENDED** that DOTS servers return a Max-Age Option in GET responses. Considerations related to which value to use and how such a value is set are implementation and deployment specific.

If an Observe Option set to 0 is included in the configuration request, the DOTS server sends notifications of any configuration change (Section 4.2 of [RFC7641]).

If a DOTS server detects that a misbehaving DOTS client does not contact the DOTS server after the expiry of Max-Age to retrieve the signal channel configuration data, it **MAY** terminate the (D)TLS session. A (D)TLS session is terminated by the receipt of an authenticated message that closes the connection (e.g., a fatal alert (Section 6 of [RFC8446])).

4.5.4. Delete DOTS Signal Channel Session Configuration

A DELETE request is used to delete the installed DOTS signal channel session configuration data (Figure 24).

```
Header: DELETE (Code=0.04)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "config"
Uri-Path: "sid=123"
```

Figure 24: Delete Configuration

The DOTS server resets the DOTS signal channel session configuration back to the default values and acknowledges a DOTS client's request to remove the DOTS signal channel session configuration using a 2.02 (Deleted) Response Code.

Upon bootstrapping or reboot, a DOTS client **MAY** send a DELETE request to set the configuration parameters to default values. Such a request does not include any 'sid'.

4.6. Redirected Signaling

Redirected DOTS signaling is discussed in detail in Section 3.2.2 of [RFC8811].

To redirect a DOTS client to an alternative DOTS server, the DOTS server can return the error Response Code 5.03 (Service Unavailable) in response to a request from the DOTS client or convey the error Response Code 5.03 in a unidirectional notification response to the client.

The DOTS server in the error response conveys the alternate DOTS server's FQDN, and the alternate DOTS server's IP address(es) values in the CBOR body (Figure 25).

```
{
  "ietf-dots-signal-channel:redirected-signal": {
    "alt-server": "string",
    "alt-server-record": [
       "string"
    ]
  }
}
```

Figure 25: Redirected Server Error Response Body Schema

The parameters are described below:

alt-server: FQDN of an alternate DOTS server.

This is a mandatory attribute.

alt-server-record: A list of IP addresses of an alternate DOTS server.

This is an optional attribute.

The DOTS server returns the Time to Live (TTL) of the alternate DOTS server in a Max-Age Option. That is, the time interval that the alternate DOTS server may be cached for use by a DOTS client. A Max- Age Option set to $2^{(32)}$-1 is equivalent to receiving an infinite TTL. This value means that the alternate DOTS server is to be used until the alternate DOTS server redirects the traffic with another 5.03 response that conveys an alternate server's FQDN.

A Max-Age Option set to '0' may be returned for redirecting mitigation requests. Such a value means that the redirection applies only for the mitigation request in progress. Returning short TTL in a Max-Age Option may adversely impact DOTS clients on slow links. Returning short values should be avoided under such conditions.

If the alternate DOTS server TTL has expired, the DOTS client **MUST** use the DOTS server(s) that was provisioned using means discussed in Section 4.1. This fallback mechanism is triggered immediately upon expiry of the TTL, except when a DDoS attack is active.

Requests issued by misbehaving DOTS clients that do not honor the TTL conveyed in the Max-Age Option or react to explicit redirect messages **MAY** be rejected by DOTS servers.

Figure 26 shows a 5.03 response example to convey the DOTS alternate server 'alt-server.example' together with its IP addresses 2001:db8:6401::1 and 2001:db8:6401::2.

```
{
  "ietf-dots-signal-channel:redirected-signal": {
    "alt-server": "alt-server.example",
    "alt-server-record": [
       "2001:db8:6401::1",
       "2001:db8:6401::2"
    ]
  }
}
```

Figure 26: Example of Redirected Server Error Response Body

When the DOTS client receives a 5.03 response with an alternate server included, it considers the current request to have failed, but it **SHOULD** try resending the request to the alternate DOTS server. During a DDoS attack, the DNS server may be the target of another DDoS attack; the alternate DOTS server's IP addresses conveyed in the 5.03 response help the DOTS client skip the DNS lookup of the alternate DOTS server, at the cost of trusting the first DOTS server to provide accurate information. The DOTS client can then try to establish a UDP or a TCP session with the alternate DOTS server (Section 4.3). Note that state synchronization (e.g., signal session configuration, aliases) is assumed to be in place between the original and alternate DOTS servers;

such synchronization means are out of scope. If session configuration refresh is needed while redirection is in place, the DOTS client follows the procedure defined in Section 4.5.3. In 'idle' time and under some conditions (e.g., infinite configuration lifetime, infinite redirection TTL, and failure to refresh the configuration), the DOTS client follows the procedure defined in Section 4.5.2 to negotiate the DOTS signal channel session configuration with the alternate server. The DOTS client **MAY** implement a method to construct IPv4-embedded IPv6 addresses [RFC6052]; this is required to handle the scenario where an IPv6-only DOTS client communicates with an IPv4-only alternate DOTS server.

If the DOTS client has been redirected to a DOTS server with which it has already communicated within the last five (5) minutes, it **MUST** ignore the redirection and try to contact other DOTS servers listed in the local configuration or discovered using dynamic means, such as DHCP or SRV procedures [RFC8973]. It is **RECOMMENDED** that DOTS clients support the means to alert administrators about redirect loops.

4.7. Heartbeat Mechanism

To provide an indication of signal health and to distinguish an 'idle' signal channel from a 'disconnected' or 'defunct' session, the DOTS agent sends a heartbeat over the signal channel to maintain its half of the channel (also, aligned with the "consents" recommendation in Section 6 of [RFC8085]). The DOTS agent similarly expects a heartbeat from its peer DOTS agent, and it may consider a session terminated in the prolonged absence of a peer agent heartbeat. Concretely, while the communication between the DOTS agents is otherwise quiescent, the DOTS client will probe the DOTS server to ensure it has maintained cryptographic state and vice versa. Such probes can also keep the bindings of firewalls and/or stateful translators alive. This probing reduces the frequency of establishing a new handshake when a DOTS signal needs to be conveyed to the DOTS server.

> Implementation Note: Given that CoAP roles can be multiplexed over the same session as discussed in [RFC7252] and are already supported by CoAP implementations, both the DOTS client and server can send DOTS heartbeat requests.

The DOTS heartbeat mechanism uses Non-confirmable PUT requests (Figure 27) with an expected 2.04 (Changed) Response Code (Figure 28). This procedure occurs between a DOTS agent and its immediate peer DOTS agent. As such, this PUT request **MUST NOT** be relayed by a DOTS gateway. The PUT request used for DOTS heartbeat **MUST NOT** have a 'cuid', 'cdid', or 'mid' Uri-Path.

```
Header: PUT (Code=0.03)
Uri-Path: ".well-known"
Uri-Path: "dots"
Uri-Path: "hb"
Content-Format: "application/dots+cbor"

{
  "ietf-dots-signal-channel:heartbeat": {
     "peer-hb-status": true
   }
}
```

Figure 27: PUT to Check Peer DOTS Agent Is Responding

The mandatory 'peer-hb-status' attribute is set to 'true' (or 'false') to indicate that a DOTS agent is (or is not) receiving heartbeat messages from its peer in the last (2 * 'heartbeat- interval') period. Such information can be used by a peer DOTS agent to detect or confirm connectivity issues and react accordingly. For example, if a DOTS client receives a 2.04 response for its heartbeat messages but no server-initiated heartbeat messages, the DOTS client sets 'peer-hb-status' to 'false' in its next heartbeat message. Upon receipt of this message, the DOTS server then will need to try another strategy for sending the heartbeats (e.g., adjust the heartbeat interval or send a server-initiated heartbeat immediately after receiving a client-initiated heartbeat message).

```
Header: (Code=2.04)
```

Figure 28: Response to a DOTS Heartbeat Request (with an Empty Body)

DOTS servers **MAY** trigger their heartbeat requests immediately after receiving heartbeat probes from peer DOTS clients. It is the responsibility of DOTS clients to ensure that on-path translators/ firewalls are maintaining a binding so that the same external IP address and/or port number is retained for the DOTS signal channel session.

Under normal traffic conditions (i.e., no attack is ongoing), if a DOTS agent does not receive any response from the peer DOTS agent for 'missing-hb-allowed' number of consecutive heartbeat messages, it concludes that the DOTS signal channel session is disconnected. The DOTS client **MUST** then try to reestablish the DOTS signal channel session, preferably by resuming the (D)TLS session.

> Note: If a new DOTS signal channel session cannot be established, the DOTS client **SHOULD NOT** retry to establish the DOTS signal channel session more frequently than every 300 seconds (5 minutes) and **MUST NOT** retry more frequently than every 60 seconds (1 minute). It is recommended that DOTS clients support the means to alert administrators about the failure to establish a (D)TLS session.

In case of a massive DDoS attack that saturates the incoming link(s) to the DOTS client, all traffic from the DOTS server to the DOTS client will likely be dropped, although the DOTS server receives heartbeat requests in addition to DOTS messages sent by the DOTS client. In this scenario, DOTS clients **MUST** behave differently to handle message transmission and DOTS signal channel session liveliness during link saturation:

> The DOTS client **MUST NOT** consider the DOTS signal channel session terminated even after a maximum 'missing-hb-allowed' threshold is reached. The DOTS client **SHOULD** keep on using the current DOTS signal channel session to send heartbeat requests over it so that the DOTS server knows the DOTS client has not disconnected the DOTS signal channel session.
>
> After the maximum 'missing-hb-allowed' threshold is reached, the DOTS client **SHOULD** try to establish a new DOTS signal channel session. The DOTS client **SHOULD** send mitigation requests over the current DOTS signal channel session and, in parallel, send the mitigation requests over the new DOTS signal channel session. This may be handled, for example, by resumption of the (D)TLS session or using 0-RTT mode in DTLS 1.3 to piggyback the mitigation request in the ClientHello message.
>
> As soon as the link is no longer saturated, if traffic from the DOTS server reaches the DOTS client over the current DOTS signal channel session, the DOTS client can stop the new DOTS signal channel session attempt or if a new DOTS signal channel session is successful then disconnect the current DOTS signal channel session.

If the DOTS server receives traffic from the peer DOTS client (e.g., peer DOTS client-initiated heartbeats) but the maximum 'missing-hb- allowed' threshold is reached, the DOTS server **MUST NOT** consider the DOTS signal channel session disconnected. The DOTS server **MUST** keep on using the current DOTS signal channel session so that the DOTS client can send mitigation requests over the current DOTS signal channel session. In this case, the DOTS server can identify that the DOTS client is under attack and that the inbound link to the DOTS client (domain) is saturated. Furthermore, if the DOTS server does not receive a mitigation request from the DOTS client, it implies that the DOTS client has not detected the attack or, if an attack mitigation is in progress, it implies that the applied DDoS mitigation actions are not yet effectively handling the DDoS attack volume.

If the DOTS server does not receive any traffic from the peer DOTS client during the time span required to exhaust the maximum 'missing-hb-allowed' threshold, the DOTS server concludes the session is disconnected. The DOTS server can then trigger preconfigured mitigation requests for this DOTS client (if any).

In DOTS over TCP, the sender of a DOTS heartbeat message has to allow up to 'heartbeat-interval' seconds when waiting for a heartbeat reply. When a failure is detected by a DOTS client, it proceeds with the session recovery, following the same approach as the one used for unreliable transports.

5. DOTS Signal Channel YANG Modules

This document defines a YANG module [RFC7950] for DOTS mitigation scope, DOTS signal channel session configuration data, DOTS redirection signaling, and DOTS heartbeats.

This YANG module is not intended to be used via NETCONF/RESTCONF for DOTS server management purposes; such a module is out of the scope of this document. It serves only to provide abstract data structures. This document uses the "structure" extension specified in [RFC8791].

A companion YANG module is defined to include a collection of types defined by IANA: "iana-dots-signal-channel" (Section 5.2).

5.1. Tree Structure

This document defines the YANG module "ietf-dots-signal-channel", which has the following tree structure. A DOTS signal message can be a mitigation, a configuration, a redirect, or a heartbeat message.

This tree structure obsoletes the one described in Section 5.1 of [RFC8782].

```
module: ietf-dots-signal-channel

  structure dots-signal:
    +-- (message-type)?
       +--:(mitigation-scope)
       |  +-- scope* []
       |     +-- target-prefix*                inet:ip-prefix
       |     +-- target-port-range* [lower-port]
       |     |  +-- lower-port    inet:port-number
       |     |  +-- upper-port?   inet:port-number
       |     +-- target-protocol*              uint8
       |     +-- target-fqdn*                  inet:domain-name
       |     +-- target-uri*                   inet:uri
       |     +-- alias-name*                   string
       |     +-- lifetime?                     union
       |     +-- trigger-mitigation?           boolean
       |     +-- (direction)?
       |        +--:(server-to-client-only)
       |        |  +-- mid?                    uint32
       |        |  +-- mitigation-start?       uint64
       |        |  +-- status?
       |        |  |       iana-dots-signal:status
       |        |  +-- conflict-information
       |        |  |  +-- conflict-status?
       |        |  |  |       iana-dots-signal:conflict-status
       |        |  |  +-- conflict-cause?
       |        |  |  |       iana-dots-signal:conflict-cause
       |        |  |  +-- retry-timer?       uint32
       |        |  |  +-- conflict-scope
       |        |  |     +-- target-prefix*       inet:ip-prefix
       |        |  |     +-- target-port-range* [lower-port]
       |        |  |     |  +-- lower-port    inet:port-number
       |        |  |     |  +-- upper-port?   inet:port-number
       |        |  |     +-- target-protocol*     uint8
       |        |  |     +-- target-fqdn*         inet:domain-name
       |        |  |     +-- target-uri*          inet:uri
       |        |  |     +-- alias-name*          string
       |        |  |     +-- acl-list* [acl-name]
       |        |  |     |  +-- acl-name    leafref
       |        |  |     |  +-- acl-type?   leafref
       |        |  |     +-- mid?                 uint32
       |        |  +-- bytes-dropped?
       |        |  |       yang:zero-based-counter64
       |        |  +-- bps-dropped?            yang:gauge64
       |        |  +-- pkts-dropped?
       |        |  |       yang:zero-based-counter64
       |        |  +-- pps-dropped?            yang:gauge64
       |        +--:(client-to-server-only)
       |           +-- attack-status?
       |                  iana-dots-signal:attack-status
       +--:(signal-config)
       |  +-- mitigating-config
       |  |  +-- heartbeat-interval
       |  |  |  +-- (direction)?
       |  |  |  |  +--:(server-to-client-only)
       |  |  |  |     +-- max-value?   uint16
       |  |  |  |     +-- min-value?   uint16
```

```
        |  |  |  +-- current-value?     uint16
        |  |  +-- missing-hb-allowed
        |  |  |  +-- (direction)?
        |  |  |  |  +--:(server-to-client-only)
        |  |  |  |     +-- max-value?   uint16
        |  |  |  |     +-- min-value?   uint16
        |  |  |  +-- current-value?     uint16
        |  |  +-- probing-rate
        |  |  |  +-- (direction)?
        |  |  |  |  +--:(server-to-client-only)
        |  |  |  |     +-- max-value?   uint16
        |  |  |  |     +-- min-value?   uint16
        |  |  |  +-- current-value?     uint16
        |  |  +-- max-retransmit
        |  |  |  +-- (direction)?
        |  |  |  |  +--:(server-to-client-only)
        |  |  |  |     +-- max-value?   uint16
        |  |  |  |     +-- min-value?   uint16
        |  |  |  +-- current-value?     uint16
        |  |  +-- ack-timeout
        |  |  |  +-- (direction)?
        |  |  |  |  +--:(server-to-client-only)
        |  |  |  |     +-- max-value-decimal?   decimal64
        |  |  |  |     +-- min-value-decimal?   decimal64
        |  |  |  +-- current-value-decimal?     decimal64
        |  |  +-- ack-random-factor
        |  |     +-- (direction)?
        |  |     |  +--:(server-to-client-only)
        |  |     |     +-- max-value-decimal?   decimal64
        |  |     |     +-- min-value-decimal?   decimal64
        |  |     +-- current-value-decimal?     decimal64
        |  +-- idle-config
        |     +-- heartbeat-interval
        |     |  +-- (direction)?
        |     |  |  +--:(server-to-client-only)
        |     |  |     +-- max-value?   uint16
        |     |  |     +-- min-value?   uint16
        |     |  +-- current-value?     uint16
        |     +-- missing-hb-allowed
        |     |  +-- (direction)?
        |     |  |  +--:(server-to-client-only)
        |     |  |     +-- max-value?   uint16
        |     |  |     +-- min-value?   uint16
        |     |  +-- current-value?     uint16
        |     +-- probing-rate
        |     |  +-- (direction)?
        |     |  |  +--:(server-to-client-only)
        |     |  |     +-- max-value?   uint16
        |     |  |     +-- min-value?   uint16
        |     |  +-- current-value?     uint16
        |     +-- max-retransmit
        |     |  +-- (direction)?
        |     |  |  +--:(server-to-client-only)
        |     |  |     +-- max-value?   uint16
        |     |  |     +-- min-value?   uint16
        |     |  +-- current-value?     uint16
        |     +-- ack-timeout
        |     |  +-- (direction)?
```

```
        |     |  |  +--:(server-to-client-only)
        |     |  |     +-- max-value-decimal?   decimal64
        |     |  |     +-- min-value-decimal?   decimal64
        |     |  +-- current-value-decimal?     decimal64
        |     +-- ack-random-factor
        |        +-- (direction)?
        |        |  +--:(server-to-client-only)
        |        |     +-- max-value-decimal?   decimal64
        |        |     +-- min-value-decimal?   decimal64
        |        +-- current-value-decimal?     decimal64
        +--:(redirected-signal)
        |  +-- (direction)?
        |     +--:(server-to-client-only)
        |        +-- alt-server           inet:domain-name
        |        +-- alt-server-record*   inet:ip-address
        +--:(heartbeat)
           +-- peer-hb-status            boolean
```

5.2. IANA DOTS Signal Channel YANG Module

This version obsoletes the version described in Section 5.2 of [RFC8782].

```
<CODE BEGINS> file "iana-dots-signal-channel@2021-09-02.yang"


module iana-dots-signal-channel {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:iana-dots-signal-channel";
  prefix iana-dots-signal;

  organization
    "IANA";
  contact
    "Internet Assigned Numbers Authority

     Postal: ICANN
          12025 Waterfront Drive, Suite 300
          Los Angeles, CA  90094-2536
          United States of America
     Tel:    +1 310 301 5800
     <mailto:iana@iana.org>";
  description
    "This module contains a collection of YANG data types defined
     by IANA and used for DOTS signal channel protocol.

     Copyright (c) 2021 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject
     to the license terms contained in, the Simplified BSD License
     set forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (http://trustee.ietf.org/license-info).

     This version of this YANG module is part of RFC 9132; see
     the RFC itself for full legal notices.";

  revision 2021-09-02 {
    description
      "Updated the prefix used for the module.";
    reference
      "RFC 9132: Distributed Denial-of-Service Open Threat
                 Signaling (DOTS) Signal Channel Specification";
  }

  revision 2020-05-28 {
    description
      "Initial revision.";
    reference
      "RFC 8782: Distributed Denial-of-Service Open Threat
                 Signaling (DOTS) Signal Channel Specification";
  }

  typedef status {
    type enumeration {
      enum attack-mitigation-in-progress {
        value 1;
        description
```

```
          "Attack mitigation setup is in progress (e.g., changing
           the network path to reroute the inbound traffic
           to DOTS mitigator).";
      }
      enum attack-successfully-mitigated {
        value 2;
        description
          "Attack is being successfully mitigated (e.g., traffic
           is redirected to a DDoS mitigator and attack
           traffic is dropped).";
      }
      enum attack-stopped {
        value 3;
        description
          "Attack has stopped and the DOTS client can
           withdraw the mitigation request.";
      }
      enum attack-exceeded-capability {
        value 4;
        description
          "Attack has exceeded the mitigation provider
           capability.";
      }
      enum dots-client-withdrawn-mitigation {
        value 5;
        description
          "DOTS client has withdrawn the mitigation
           request and the mitigation is active but
           terminating.";
      }
      enum attack-mitigation-terminated {
        value 6;
        description
          "Attack mitigation is now terminated.";
      }
      enum attack-mitigation-withdrawn {
        value 7;
        description
          "Attack mitigation is withdrawn.";
      }
      enum attack-mitigation-signal-loss {
        value 8;
        description
          "Attack mitigation will be triggered
           for the mitigation request only when
           the DOTS signal channel session is lost.";
      }
    }
    description
      "Enumeration for status reported by the DOTS server.";
  }

  typedef conflict-status {
    type enumeration {
      enum request-inactive-other-active {
        value 1;
        description
          "DOTS server has detected conflicting mitigation
```

```
            requests from different DOTS clients.
            This mitigation request is currently inactive
            until the conflicts are resolved.  Another
            mitigation request is active.";
        }
        enum request-active {
          value 2;
          description
            "DOTS server has detected conflicting mitigation
             requests from different DOTS clients.
             This mitigation request is currently active.";
        }
        enum all-requests-inactive {
          value 3;
          description
            "DOTS server has detected conflicting mitigation
             requests from different DOTS clients.  All
             conflicting mitigation requests are inactive.";
        }
      }
      description
        "Enumeration for conflict status.";
    }

    typedef conflict-cause {
      type enumeration {
        enum overlapping-targets {
          value 1;
          description
            "Overlapping targets. conflict-scope provides
             more details about the exact conflict.";
        }
        enum conflict-with-acceptlist {
          value 2;
          description
            "Conflicts with an existing accept-list.

             This code is returned when the DDoS mitigation
             detects that some of the source addresses/prefixes
             listed in the accept-list ACLs are actually
             attacking the target.";
        }
        enum cuid-collision {
          value 3;
          description
            "Conflicts with the cuid used by another
             DOTS client.";
        }
      }
      description
        "Enumeration for conflict causes.";
    }

    typedef attack-status {
      type enumeration {
        enum under-attack {
          value 1;
          description
```

```
            "The DOTS client determines that it is still under
             attack.";
        }
        enum attack-successfully-mitigated {
          value 2;
          description
            "The DOTS client determines that the attack is
             successfully mitigated.";
        }
      }
      description
        "Enumeration for attack status codes.";
    }
  }

<CODE ENDS>
```

5.3. IETF DOTS Signal Channel YANG Module

This module uses the common YANG types defined in [RFC6991] and types defined in [RFC8783].

This version obsoletes the version described in Section 5.3 of [RFC8782].

```
<CODE BEGINS> file "ietf-dots-signal-channel@2021-09-02.yang"


module ietf-dots-signal-channel {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-dots-signal-channel";
  prefix dots-signal;

  import ietf-inet-types {
    prefix inet;
    reference
      "Section 4 of RFC 6991";
  }
  import ietf-yang-types {
    prefix yang;
    reference
      "Section 3 of RFC 6991";
  }
  import ietf-dots-data-channel {
    prefix data-channel;
    reference
      "RFC 8783: Distributed Denial-of-Service Open Threat Signaling
                 (DOTS) Data Channel Specification";
  }
  import iana-dots-signal-channel {
    prefix iana-dots-signal;
    reference
      "RFC 9132: Distributed Denial-of-Service Open Threat Signaling
                 (DOTS) Signal Channel Specification";
  }
  import ietf-yang-structure-ext {
    prefix sx;
    reference
      "RFC 8791: YANG Data Structure Extensions";
  }

  organization
    "IETF DDoS Open Threat Signaling (DOTS) Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/dots/>
     WG List:  <mailto:dots@ietf.org>

     Editor:  Mohamed Boucadair
              <mailto:mohamed.boucadair@orange.com>

     Editor:  Jon Shallow
              <mailto:supjps-ietf@jpshallow.com>

     Author:  Konda, Tirumaleswar Reddy.K
              <mailto:kondtir@gmail.com>

     Author:  Prashanth Patil
              <mailto:praspati@cisco.com>

     Author:  Andrew Mortensen
              <mailto:amortensen@arbor.net>
```

```
      Author:  Nik Teague
               <mailto:nteague@ironmountain.co.uk>";
   description
     "This module contains YANG definition for the signaling
      messages exchanged between a DOTS client and a DOTS server.

      Copyright (c) 2021 IETF Trust and the persons identified as
      authors of the code.  All rights reserved.

      Redistribution and use in source and binary forms, with or
      without modification, is permitted pursuant to, and subject
      to the license terms contained in, the Simplified BSD License
      set forth in Section 4.c of the IETF Trust's Legal Provisions
      Relating to IETF Documents
      (http://trustee.ietf.org/license-info).

      This version of this YANG module is part of RFC 9132; see
      the RFC itself for full legal notices.";

   revision 2021-09-02 {
     description
       "Updated revision to comply with RFC 8791.

        This version is not backward compatible with the version
        published in RFC 8782.";
     reference
       "RFC 9132: Distributed Denial-of-Service Open Threat
                  Signaling (DOTS) Signal Channel Specification";
   }
   revision 2020-05-28 {
     description
       "Initial revision.";
     reference
       "RFC 8782: Distributed Denial-of-Service Open Threat
                  Signaling (DOTS) Signal Channel Specification";
   }

   /*
    * Groupings
    */

   grouping mitigation-scope {
     description
       "Specifies the scope of the mitigation request.";
     list scope {
       description
         "The scope of the request.";
       uses data-channel:target;
       leaf-list alias-name {
         type string;
         description
           "An alias name that points to a resource.";
       }
       leaf lifetime {
         type union {
           type uint32;
           type int32 {
             range "-1";
```

```
          }
        }
        units "seconds";
        default "3600";
        description
          "Indicates the lifetime of the mitigation request.

           A lifetime of '0' in a mitigation request is an
           invalid value.

           A lifetime of negative one (-1) indicates indefinite
           lifetime for the mitigation request.

           Lifetime is mandatory in a mitigation request.

           The DOTS server must always indicate the actual lifetime
           in the response to an accepted mitigation request and the
           remaining lifetime in status messages sent to the
           DOTS client.";
      }
      leaf trigger-mitigation {
        type boolean;
        default "true";
        description
          "If set to 'false', DDoS mitigation will not be
           triggered unless the DOTS signal channel
           session is lost.";
      }
      choice direction {
        description
          "Indicates the communication direction in which the
           data nodes can be included.";
        case server-to-client-only {
          description
            "These data nodes appear only in a mitigation message
             sent from the server to the client.";
          leaf mid {
            type uint32;
            description
              "Mitigation request identifier.

               This identifier must be unique for each mitigation
               request bound to the DOTS client.";
          }
          leaf mitigation-start {
            type uint64;
            description
              "Mitigation start time is represented in seconds
               relative to 1970-01-01T00:00:00Z in UTC time.

               This is a mandatory attribute when an attack
               mitigation is active.  It must not be returned for
               a mitigation with 'status' code set to 8.";
          }
          leaf status {
            type iana-dots-signal:status;
            description
              "Indicates the status of a mitigation request.
```

```
                It must be included in responses only.

                This is a mandatory attribute if a mitigation
                request is accepted and processed by the server.";
            }
            container conflict-information {
              description
                "Indicates that a conflict is detected.";
              leaf conflict-status {
                type iana-dots-signal:conflict-status;
                description
                  "Indicates the conflict status.";
              }
              leaf conflict-cause {
                type iana-dots-signal:conflict-cause;
                description
                  "Indicates the cause of the conflict.";
              }
              leaf retry-timer {
                type uint32;
                units "seconds";
                description
                  "The DOTS client must not resend the
                   same request that has a conflict before the expiry
                   of this timer.";
              }
              container conflict-scope {
                description
                  "Provides more information about the conflict
                   scope.";
                uses data-channel:target {
                  when "/dots-signal/scope/conflict-information/"
                     + "conflict-cause = 'overlapping-targets'";
                }
                leaf-list alias-name {
                  when "../../conflict-cause = 'overlapping-targets'";
                  type string;
                  description
                    "Conflicting alias-name.";
                }
                list acl-list {
                  when "../../conflict-cause ="
                     + " 'conflict-with-acceptlist'";
                  key "acl-name";
                  description
                    "List of conflicting ACLs, as defined in the DOTS
                     data channel.  These ACLs are uniquely defined by
                     cuid and acl-name.";
                  leaf acl-name {
                    type leafref {
                      path "/data-channel:dots-data"
                         + "/data-channel:dots-client"
                         + "/data-channel:acls"
                         + "/data-channel:acl/data-channel:name";
                    }
                    description
                      "Reference to the conflicting ACL name bound to
                       a DOTS client.";
```

```
            }
            leaf acl-type {
              type leafref {
                path "/data-channel:dots-data"
                   + "/data-channel:dots-client"
                   + "/data-channel:acls"
                   + "/data-channel:acl/data-channel:type";
              }
              description
                "Reference to the conflicting ACL type bound to
                 a DOTS client.";
            }
          }
          leaf mid {
            when "../../conflict-cause = 'overlapping-targets'";
            type uint32;
            description
              "Reference to the conflicting 'mid' bound to
               the same DOTS client.";
          }
        }
      }
      leaf bytes-dropped {
        type yang:zero-based-counter64;
        units "bytes";
        description
          "The total dropped byte count for the mitigation
           request since the attack mitigation was triggered.
           The count wraps around when it reaches the maximum
           value of counter64 for dropped bytes.";
      }
      leaf bps-dropped {
        type yang:gauge64;
        units "bytes per second";
        description
          "The average number of dropped bytes per second for
           the mitigation request since the attack
           mitigation was triggered.  This should be over
           five-minute intervals (that is, measuring bytes
           into five-minute buckets and then averaging these
           buckets over the time since the mitigation was
           triggered).";
      }
      leaf pkts-dropped {
        type yang:zero-based-counter64;
        description
          "The total number of dropped packet count for the
           mitigation request since the attack mitigation was
           triggered.  The count wraps around when it reaches
           the maximum value of counter64 for dropped packets.";
      }
      leaf pps-dropped {
        type yang:gauge64;
        units "packets per second";
        description
          "The average number of dropped packets per second
           for the mitigation request since the attack
           mitigation was triggered.  This should be over
```

```
               five-minute intervals (that is, measuring packets
               into five-minute buckets and then averaging these
               buckets over the time since the mitigation was
               triggered).";
          }
        }
        case client-to-server-only {
          description
            "These data nodes appear only in a mitigation message
             sent from the client to the server.";
          leaf attack-status {
            type iana-dots-signal:attack-status;
            description
              "Indicates the status of an attack as seen by the
               DOTS client.

               This is a mandatory attribute when a client
               performs an efficacy update.";
          }
        }
      }
    }
  }

  grouping config-parameters {
    description
      "Subset of DOTS signal channel session configuration.";
    container heartbeat-interval {
      description
        "DOTS agents regularly send heartbeats to each other
         after mutual authentication is successfully
         completed in order to keep the DOTS signal channel
         open.";
      choice direction {
        description
          "Indicates the communication direction in which the
           data nodes can be included.";
        case server-to-client-only {
          description
            "These data nodes appear only in a mitigation message
             sent from the server to the client.";
          leaf max-value {
            type uint16;
            units "seconds";
            description
              "Maximum acceptable heartbeat-interval value.";
          }
          leaf min-value {
            type uint16;
            units "seconds";
            description
              "Minimum acceptable heartbeat-interval value.";
          }
        }
      }
      leaf current-value {
        type uint16;
        units "seconds";
```

```
        default "30";
        description
          "Current heartbeat-interval value.

           '0' means that heartbeat mechanism is deactivated.";
      }
    }
    container missing-hb-allowed {
      description
        "Maximum number of missing heartbeats allowed.";
      choice direction {
        description
          "Indicates the communication direction in which the
           data nodes can be included.";
        case server-to-client-only {
          description
            "These data nodes appear only in a mitigation message
             sent from the server to the client.";
          leaf max-value {
            type uint16;
            description
              "Maximum acceptable missing-hb-allowed value.";
          }
          leaf min-value {
            type uint16;
            description
              "Minimum acceptable missing-hb-allowed value.";
          }
        }
      }
      leaf current-value {
        type uint16;
        default "15";
        description
          "Current missing-hb-allowed value.";
      }
    }
    container probing-rate {
      description
        "The limit for sending Non-confirmable messages with
         no response.";
      choice direction {
        description
          "Indicates the communication direction in which the
           data nodes can be included.";
        case server-to-client-only {
          description
            "These data nodes appear only in a mitigation message
             sent from the server to the client.";
          leaf max-value {
            type uint16;
            units "byte/second";
            description
              "Maximum acceptable probing-rate value.";
          }
          leaf min-value {
            type uint16;
            units "byte/second";
```

```
            description
              "Minimum acceptable probing-rate value.";
          }
        }
      }
      leaf current-value {
        type uint16;
        units "byte/second";
        default "5";
        description
          "Current probing-rate value.";
      }
    }
    container max-retransmit {
      description
        "Maximum number of retransmissions of a Confirmable
         message.";
      choice direction {
        description
          "Indicates the communication direction in which the
           data nodes can be included.";
        case server-to-client-only {
          description
            "These data nodes appear only in a mitigation message
             sent from the server to the client.";
          leaf max-value {
            type uint16;
            description
              "Maximum acceptable max-retransmit value.";
          }
          leaf min-value {
            type uint16;
            description
              "Minimum acceptable max-retransmit value.";
          }
        }
      }
      leaf current-value {
        type uint16;
        default "3";
        description
          "Current max-retransmit value.";
      }
    }
    container ack-timeout {
      description
        "Initial retransmission timeout value.";
      choice direction {
        description
          "Indicates the communication direction in which the
           data nodes can be included.";
        case server-to-client-only {
          description
            "These data nodes appear only in a mitigation message
             sent from the server to the client.";
          leaf max-value-decimal {
            type decimal64 {
              fraction-digits 2;
```

```
            }
            units "seconds";
            description
              "Maximum ack-timeout value.";
          }
          leaf min-value-decimal {
            type decimal64 {
              fraction-digits 2;
            }
            units "seconds";
            description
              "Minimum ack-timeout value.";
          }
        }
      }
      leaf current-value-decimal {
        type decimal64 {
          fraction-digits 2;
        }
        units "seconds";
        default "2";
        description
          "Current ack-timeout value.";
      }
    }
    container ack-random-factor {
      description
        "Random factor used to influence the timing of
         retransmissions.";
      choice direction {
        description
          "Indicates the communication direction in which the
           data nodes can be included.";
        case server-to-client-only {
          description
            "These data nodes appear only in a mitigation message
             sent from the server to the client.";
          leaf max-value-decimal {
            type decimal64 {
              fraction-digits 2;
            }
            description
              "Maximum acceptable ack-random-factor value.";
          }
          leaf min-value-decimal {
            type decimal64 {
              fraction-digits 2;
            }
            description
              "Minimum acceptable ack-random-factor value.";
          }
        }
      }
      leaf current-value-decimal {
        type decimal64 {
          fraction-digits 2;
        }
        default "1.5";
```

```
        description
          "Current ack-random-factor value.";
      }
    }
  }

  grouping signal-config {
    description
      "DOTS signal channel session configuration.";
    container mitigating-config {
      description
        "Configuration parameters to use when a mitigation
         is active.";
      uses config-parameters;
    }
    container idle-config {
      description
        "Configuration parameters to use when no mitigation
         is active.";
      uses config-parameters;
    }
  }

  grouping redirected-signal {
    description
      "Grouping for the redirected signaling.";
    choice direction {
      description
        "Indicates the communication direction in which the
         data nodes can be included.";
      case server-to-client-only {
        description
          "These data nodes appear only in a mitigation message
           sent from the server to the client.";
        leaf alt-server {
          type inet:domain-name;
          mandatory true;
          description
            "FQDN of an alternate server.";
        }
        leaf-list alt-server-record {
          type inet:ip-address;
          description
            "List of records for the alternate server.";
        }
      }
    }
  }

  /*
   * DOTS Signal Channel Structure
   */

  sx:structure dots-signal {
    description
      "Main structure for DOTS signal message.

       A DOTS signal message can be a mitigation, a configuration,
```

```
        a redirected, or a heartbeat signal message.";
    choice message-type {
      description
        "Can be a mitigation, a configuration, a redirect, or
         a heartbeat message.";
      case mitigation-scope {
        description
          "Mitigation scope of a mitigation message.";
        uses mitigation-scope;
      }
      case signal-config {
        description
          "Configuration message.";
        uses signal-config;
      }
      case redirected-signal {
        description
          "Redirected signaling.";
        uses redirected-signal;
      }
      case heartbeat {
        description
          "DOTS heartbeats.";
        leaf peer-hb-status {
          type boolean;
          mandatory true;
          description
            "Indicates whether a DOTS agent receives heartbeats
             from its peer.  The value is set to 'true' if the
             DOTS agent is receiving heartbeat messages
             from its peer.";
        }
      }
    }
  }
}

<CODE ENDS>
```

6. YANG/JSON Mapping Parameters to CBOR

All parameters in the payload of the DOTS signal channel **MUST** be mapped to CBOR types, as shown in Table 5, and are assigned an integer key to save space.

> Note: Implementers must check that the mapping output provided by their YANG-to-CBOR encoding schemes is aligned with the content of Table 5. For example, some CBOR and JSON types for enumerations and the 64-bit quantities can differ depending on the encoder used.

The CBOR key values are divided into two types: comprehension-required and comprehension-optional. DOTS agents can safely ignore comprehension-optional values they don't understand, but they cannot successfully process a request if it contains comprehension-required values that

are not understood. The 4.00 response **SHOULD** include a diagnostic payload describing the unknown comprehension-required CBOR key values. The initial set of CBOR key values defined in this specification are of type comprehension-required.

Parameter Name	YANG Type	CBOR Key	CBOR Major Type & Information	JSON Type
ietf-dots-signal-channel:mitigation-scope	container	1	5 map	Object
scope	list	2	4 array	Array
cdid	string	3	3 text string	String
cuid	string	4	3 text string	String
mid	uint32	5	0 unsigned	Number
target-prefix	leaf-list	6	4 array	Array
	inet:ip-prefix		3 text string	String
target-port-range	list	7	4 array	Array
lower-port	inet:port-number	8	0 unsigned	Number
upper-port	inet:port-number	9	0 unsigned	Number
target-protocol	leaf-list	10	4 array	Array
	uint8		0 unsigned	Number
target-fqdn	leaf-list	11	4 array	Array
	inet:domain-name		3 text string	String
target-uri	leaf-list	12	4 array	Array
	inet:uri		3 text string	String
alias-name	leaf-list	13	4 array	Array
	string		3 text string	String
lifetime	union	14	0 unsigned	Number
			1 negative	Number

Parameter Name	YANG Type	CBOR Key	CBOR Major Type & Information	JSON Type
mitigation-start	uint64	15	0 unsigned	String
status	enumeration	16	0 unsigned	String
conflict-information	container	17	5 map	Object
conflict-status	enumeration	18	0 unsigned	String
conflict-cause	enumeration	19	0 unsigned	String
retry-timer	uint32	20	0 unsigned	String
conflict-scope	container	21	5 map	Object
acl-list	list	22	4 array	Array
acl-name	leafref	23	3 text string	String
acl-type	leafref	24	3 text string	String
bytes-dropped	yang:zero-based-counter64	25	0 unsigned	String
bps-dropped	yang:gauge64	26	0 unsigned	String
pkts-dropped	yang:zero-based-counter64	27	0 unsigned	String
pps-dropped	yang:gauge64	28	0 unsigned	String
attack-status	enumeration	29	0 unsigned	String
ietf-dots-signal-channel:signal-config	container	30	5 map	Object
sid	uint32	31	0 unsigned	Number
mitigating-config	container	32	5 map	Object
heartbeat-interval	container	33	5 map	Object
max-value	uint16	34	0 unsigned	Number
min-value	uint16	35	0 unsigned	Number
current-value	uint16	36	0 unsigned	Number

Parameter Name	YANG Type	CBOR Key	CBOR Major Type & Information	JSON Type
missing-hb-allowed	container	37	5 map	Object
max-retransmit	container	38	5 map	Object
ack-timeout	container	39	5 map	Object
ack-random-factor	container	40	5 map	Object
max-value-decimal	decimal64	41	6 tag 4 [-2, integer]	String
min-value-decimal	decimal64	42	6 tag 4 [-2, integer]	String
current-value-decimal	decimal64	43	6 tag 4 [-2, integer]	String
idle-config	container	44	5 map	Object
trigger-mitigation	boolean	45	7 bits 20	False
			7 bits 21	True
ietf-dots-signal-channel:redirected-signal	container	46	5 map	Object
alt-server	inet:domain-name	47	3 text string	String
alt-server-record	leaf-list	48	4 array	Array
	inet:ip-address		3 text string	String
ietf-dots-signal-channel:heartbeat	container	49	5 map	Object
probing-rate	container	50	5 map	Object
peer-hb-status	boolean	51	7 bits 20	False
			7 bits 21	True

Table 5: CBOR Key Values Used in DOTS Signal Channel Messages & Their Mappings to JSON and YANG

7. (D)TLS Protocol Profile and Performance Considerations

7.1. (D)TLS Protocol Profile

This section defines the (D)TLS protocol profile of DOTS signal channel over (D)TLS and DOTS data channel over TLS.

There are known attacks on (D)TLS, such as man-in-the-middle and protocol downgrade attacks. These are general attacks on (D)TLS and, as such, they are not specific to DOTS over (D)TLS; refer to the (D)TLS RFCs for discussion of these security issues. DOTS agents **MUST** adhere to the (D)TLS implementation recommendations and security considerations of [RFC7525] except with respect to (D)TLS version. Because DOTS signal channel encryption relying upon (D)TLS is virtually a greenfield deployment, DOTS agents **MUST** implement only (D)TLS 1.2 or later.

When a DOTS client is configured with a domain name of the DOTS server, and it connects to its configured DOTS server, the server may present it with a PKIX certificate. In order to ensure proper authentication, a DOTS client **MUST** verify the entire certification path per [RFC5280]. Additionally, the DOTS client **MUST** use [RFC6125] validation techniques to compare the domain name with the certificate provided. Certification authorities that issue DOTS server certificates **SHOULD** support the DNS-ID and SRV-ID identifier types. DOTS servers **SHOULD** prefer the use of DNS-ID and SRV-ID over Common Name ID (CN-ID) identifier types in certificate requests (as described in Section 2.3 of [RFC6125]), and the wildcard character '*' **SHOULD NOT** be included in the presented identifier. DOTS doesn't use URI-IDs for server identity verification.

A key challenge to deploying DOTS is the provisioning of DOTS clients, including the distribution of keying material to DOTS clients to enable the required mutual authentication of DOTS agents. Enrollment over Secure Transport (EST) [RFC7030] defines a method of certificate enrollment by which domains operating DOTS servers may provide DOTS clients with all the necessary cryptographic keying material, including a private key and a certificate, to authenticate themselves. One deployment option is to have DOTS clients behave as EST clients for certificate enrollment from an EST server provisioned by the mitigation provider. This document does not specify which EST or other mechanism the DOTS client uses to achieve initial enrollment.

The Server Name Indication (SNI) extension [RFC6066] defines a mechanism for a client to tell a (D)TLS server the name of the server it wants to contact. This is a useful extension for hosting environments where multiple virtual servers are reachable over a single IP address. The DOTS client may or may not know if it is interacting with a DOTS server in a virtual server-hosting environment, so the DOTS client **SHOULD** include the DOTS server FQDN in the SNI extension.

Implementations compliant with this profile **MUST** implement all of the following items:

- DTLS record replay detection (Section 3.3 of [RFC6347]) or an equivalent mechanism to protect against replay attacks.
- DTLS session resumption without server-side state to resume session and convey the DOTS signal.

- At least one of raw public keys [RFC7250] or PSK handshake [RFC4279] with (EC)DHE key exchange. This reduces the size of the ServerHello. Also, this can be used by DOTS agents that cannot obtain certificates.

Implementations compliant with this profile **SHOULD** implement all of the following items to reduce the delay required to deliver a DOTS signal channel message:

- TLS False Start [RFC7918], which reduces round trips by allowing the TLS client's second flight of messages (ChangeCipherSpec) to also contain the DOTS signal. TLS False Start is formally defined for use with TLS, but the same technique is applicable to DTLS as well.
- Cached Information Extension [RFC7924], which avoids transmitting the server's certificate and certificate chain if the client has cached that information from a previous TLS handshake.

Compared to UDP, DOTS signal channel over TCP requires an additional round-trip time (RTT) of latency to establish a TCP connection. DOTS implementations are encouraged to implement TCP Fast Open [RFC7413] to eliminate that RTT.

7.2. (D)TLS 1.3 Considerations

TLS 1.3 provides useful latency improvements for connection establishment over TLS 1.2. The DTLS 1.3 protocol [TLS-DTLS13] is based upon the TLS 1.3 protocol and provides equivalent security guarantees. (D)TLS 1.3 provides two basic handshake modes the DOTS signal channel can take advantage of:

- A full handshake mode in which a DOTS client can send a DOTS mitigation request message after one round trip and the DOTS server immediately responds with a DOTS mitigation response. This assumes no packet loss is experienced.
- 0-RTT mode in which the DOTS client can authenticate itself and send DOTS mitigation request messages in the first message, thus reducing handshake latency. 0-RTT only works if the DOTS client has previously communicated with that DOTS server, which is very likely with the DOTS signal channel.

The DOTS client has to establish a (D)TLS session with the DOTS server during 'idle' time and share a PSK.

During a DDoS attack, the DOTS client can use the (D)TLS session to convey the DOTS mitigation request message and, if there is no response from the server after multiple retries, the DOTS client can resume the (D)TLS session in 0-RTT mode using PSK.

DOTS servers that support (D)TLS 1.3 **MAY** allow DOTS clients to send early data (0-RTT). DOTS clients **MUST NOT** send "CoAP Ping" as early data; such messages **MUST** be rejected by DOTS servers. Section 8 of [RFC8446] discusses some mechanisms to implement in order to limit the impact of replay attacks on 0-RTT data. If the DOTS server accepts 0-RTT, it **MUST** implement one of these mechanisms to prevent replay at the TLS layer. A DOTS server can reject 0-RTT by sending a TLS HelloRetryRequest.

The DOTS signal channel messages sent as early data by the DOTS client are idempotent requests. As a reminder, the Message ID (Section 3 of [RFC7252]) is changed each time a new CoAP request is sent, and the Token (Section 5.3.1 of [RFC7252]) is randomized in each CoAP request. The DOTS server(s) **MUST** use the Message ID and the Token in the DOTS signal channel message to detect replay of early data at the application layer and accept 0-RTT data at most once from the same DOTS client. This anti-replay defense requires sharing the Message ID and the Token in the 0-RTT data between DOTS servers in the DOTS server domain. DOTS servers do not rely on transport coordinates to identify DOTS peers. As specified in Section 4.4.1, DOTS servers couple the DOTS signal channel sessions using the DOTS client identity and optionally the 'cdid' parameter value. Furthermore, the 'mid' value is monotonically increased by the DOTS client for each mitigation request, thus attackers that replay mitigation requests with lower numeric 'mid' values and overlapping scopes with mitigation requests having higher numeric 'mid' values will be rejected systematically by the DOTS server. Likewise, the 'sid' value is monotonically increased by the DOTS client for each configuration request (Section 4.5.2); attackers replaying configuration requests with lower numeric 'sid' values will be rejected by the DOTS server if it maintains a higher numeric 'sid' value for this DOTS client.

Owing to the aforementioned protections, all DOTS signal channel requests are safe to transmit in TLS 1.3 as early data. Refer to [DOTS-EARLYDATA] for more details.

A simplified TLS 1.3 handshake with 0-RTT DOTS mitigation request message exchange is shown in Figure 29.

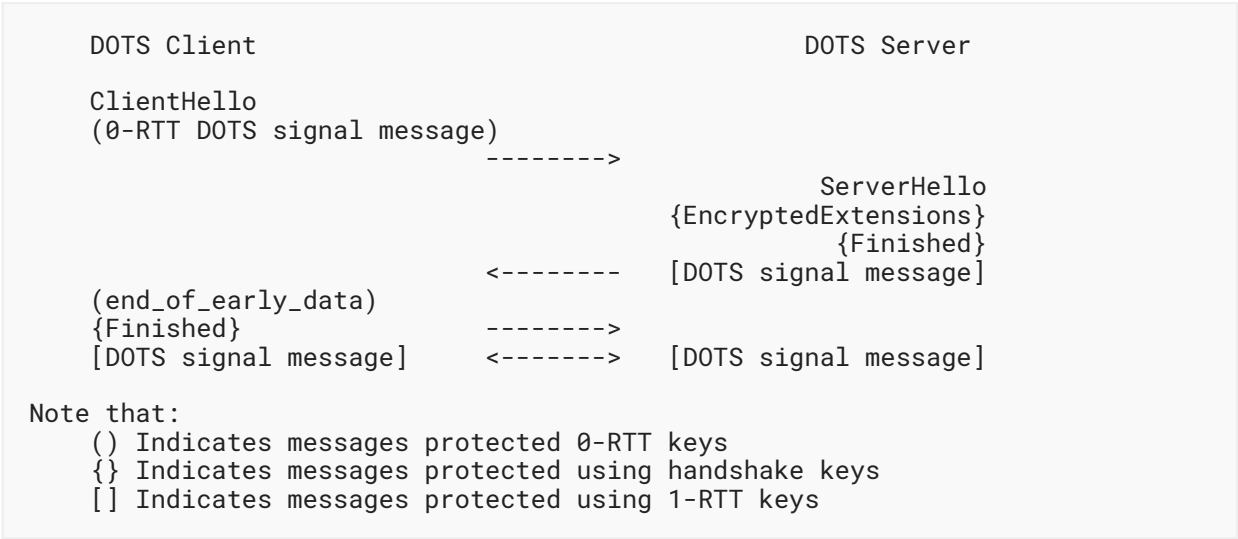

Figure 29: A Simplified TLS 1.3 Handshake with 0-RTT

7.3. DTLS MTU and Fragmentation

To avoid DOTS signal message fragmentation and the subsequent decreased probability of message delivery, the DLTS records need to fit within a single datagram [RFC6347]. DTLS handles fragmentation and reassembly only for handshake messages and not for the application data (Section 4.1.1 of [RFC6347]). If the Path MTU (PMTU) cannot be discovered, DOTS agents **MUST**

assume a PMTU of 1280 bytes, as IPv6 requires that every link in the Internet have an MTU of 1280 octets or greater, as specified in [RFC8200]. If IPv4 support on legacy or otherwise unusual networks is a consideration and the PMTU is unknown, DOTS implementations **MAY** assume a PMTU of 576 bytes for IPv4 datagrams (see Section 3.3.3 of [RFC1122]).

The DOTS client must consider the amount of record expansion expected by the DTLS processing when calculating the size of the CoAP message that fits within the PMTU. The PMTU **MUST** be greater than or equal to [CoAP message size + DTLS 1.2 overhead of 13 octets + authentication overhead of the negotiated DTLS cipher suite + block padding] (Section 4.1.1.1 of [RFC6347]). If the total request size exceeds the PMTU, then the DOTS client **MUST** split the DOTS signal into separate messages; for example, the list of addresses in the 'target-prefix' parameter could be split into multiple lists and each list conveyed in a new PUT request.

> Implementation Note: DOTS choice of message size parameters works well with IPv6 and with most of today's IPv4 paths. However, with IPv4, it is harder to safely make sure that there is no IP fragmentation. If the IPv4 PMTU is unknown, implementations may want to limit themselves to more conservative IPv4 datagram sizes, such as 576 bytes, per [RFC0791].

8. Mutual Authentication of DOTS Agents & Authorization of DOTS Clients

(D)TLS based upon client certificates can be used for mutual authentication between DOTS agents. If, for example, a DOTS gateway is involved, DOTS clients and DOTS gateways must perform mutual authentication; only authorized DOTS clients are allowed to send DOTS signals to a DOTS gateway. The DOTS gateway and the DOTS server must perform mutual authentication; a DOTS server only allows DOTS signal channel messages from an authorized DOTS gateway, thereby creating a two-link chain of transitive authentication between the DOTS client and the DOTS server.

The DOTS server should support certificate-based client authentication. The DOTS client should respond to the DOTS server's TLS CertificateRequest message with the PKIX certificate held by the DOTS client. DOTS client certificate validation must be performed per [RFC5280], and the DOTS client certificate must conform to the [RFC5280] certificate profile. If a DOTS client does not support TLS client certificate authentication, it must support client authentication based on pre-shared key or raw public key.

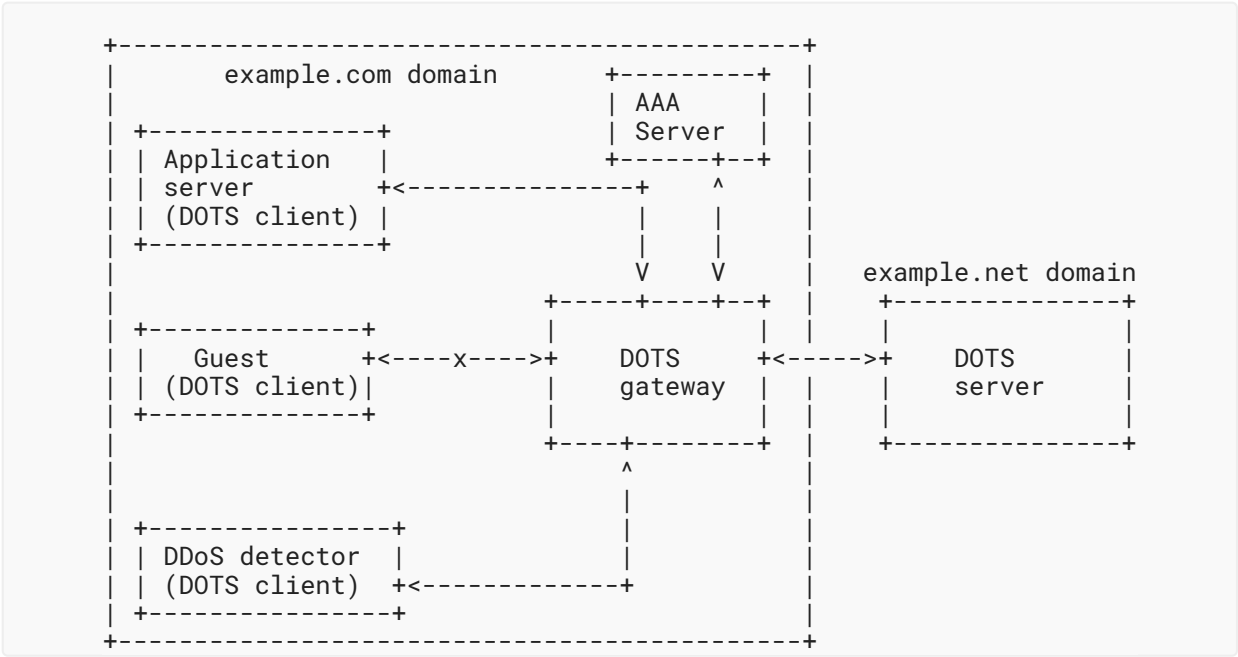

Figure 30: Example of Authentication and Authorization of DOTS Agents

In the example depicted in Figure 30, the DOTS gateway and DOTS clients within the 'example.com' domain proceed with mutual authentication. After the DOTS gateway validates the identity of a DOTS client, it communicates with the Authentication, Authorization, and Accounting (AAA) server in the 'example.com' domain to determine if the DOTS client is authorized to request DDoS mitigation. If the DOTS client is not authorized, a 4.01 (Unauthorized) is returned in the response to the DOTS client. In this example, the DOTS gateway only allows the application server and DDoS attack detector to request DDoS mitigation, but does not permit the user of type 'guest' to request DDoS mitigation.

Also, DOTS gateways and servers located in different domains must perform mutual authentication (e.g., using certificates). A DOTS server will only allow a DOTS gateway with a certificate for a particular domain to request mitigation for that domain. In reference to Figure 30, the DOTS server only allows the DOTS gateway to request mitigation for the 'example.com' domain and not for other domains.

9. Error Handling

This section is a summary of the Error Code responses that can be returned by a DOTS server. These error responses must contain a CoAP 4.xx or 5.xx Response Code.

In general, there may be an additional plain text diagnostic payload (Section 5.5.2 of [RFC7252]) to help troubleshooting in the body of the response unless detailed otherwise.

Errors returned by a DOTS server can be broken into two categories: those associated with CoAP itself and those generated during the validation of the provided data by the DOTS server.

The following is a list of common CoAP errors that are implemented by DOTS servers. This list is not exhaustive; other errors defined by CoAP and associated RFCs may be applicable.

4.00 (Bad Request) is returned by the DOTS server when the DOTS client has sent a request that violates the DOTS protocol (Section 4).

4.01 (Unauthorized) is returned by the DOTS server when the DOTS client is not authorized to access the DOTS server (Section 4).

4.02 (Bad Option) is returned by the DOTS server when one or more CoAP options are unknown or malformed by the CoAP layer [RFC7252].

4.04 (Not Found) is returned by the DOTS server when the DOTS client is requesting a 'mid' or 'sid' that is not valid (Section 4).

4.05 (Method Not Allowed) is returned by the DOTS server when the DOTS client is requesting a resource by a method (e.g., GET) that is not supported by the DOTS server [RFC7252].

4.08 (Request Entity Incomplete) is returned by the DOTS server if one or multiple blocks of a block transfer request is missing [RFC7959].

4.09 (Conflict) is returned by the DOTS server if the DOTS server detects that a request conflicts with a previous request. The response body is formatted using "application/dots+cbor" and contains the "conflict-clause" (Section 4.4.1.3).

4.13 (Request Entity Too Large) may be returned by the DOTS server during a block transfer request [RFC7959].

4.15 (Unsupported Content-Format) is returned by the DOTS server when the Content-Format is used but the request is not formatted as "application/dots+cbor" (Section 4).

4.22 (Unprocessable Entity) is returned by the DOTS server when one or more session configuration parameters are not valid (Section 4.5).

5.03 (Service Unavailable) is returned by the DOTS server if the DOTS server is unable to handle the request (Section 4). An example is the DOTS server needs to redirect the DOTS client to use an alternate DOTS server (Section 4.6). The response body is formatted using "application/dots +cbor" and contains how to handle the 5.03 Response Code.

5.08 (Hop Limit Reached) is returned by the DOTS server if there is a data path loop through upstream DOTS gateways. The response body is formatted using plain text and contains a list of servers that are in the data path loop [RFC8768].

10. IANA Considerations

10.1. DOTS Signal Channel UDP and TCP Port Number

IANA has assigned the port number 4646 (the ASCII decimal value for ".." (DOTS)) to the DOTS signal channel protocol for both UDP and TCP from the "Service Name and Transport Protocol Port Number Registry" available at <https://www.iana.org/assignments/service-names-port-numbers/>.

IANA has updated these entries to refer to this document and updated the Description as described below:

Service Name: dots-signal
Port Number: 4646
Transport Protocol: TCP
Description: Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Protocol. The service name is used to construct the SRV service names "_dots-signal._udp" and "_dots-signal._tcp" for discovering DOTS servers used to establish DOTS signal channel.
Assignee: IESG
Contact: IETF Chair
Registration Date: 2020-01-16
Reference: [RFC8973][RFC9132]

Service Name: dots-signal
Port Number: 4646
Transport Protocol: UDP
Description: Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Protocol. The service name is used to construct the SRV service names "_dots-signal._udp" and "_dots-signal._tcp" for discovering DOTS servers used to establish DOTS signal channel.
Assignee: IESG
Contact: IETF Chair
Registration Date: 2020-01-16
Reference: [RFC8973][RFC9132]

10.2. Well-Known 'dots' URI

IANA has updated the 'dots' well-known URI (Table 6) entry in the "Well-Known URIs" registry [URI] as follows:

URI Suffix	Change Controller	Reference	Status	Related information
dots	IETF	[RFC9132]	permanent	None

Table 6: 'dots' Well-Known URI

10.3. Media Type Registration

IANA has updated the "application/dots+cbor" media type in the "Media Types" registry [IANA-MediaTypes] in the manner described in [RFC6838], which can be used to indicate that the content is a DOTS signal channel object:

Type name: application

Subtype name: dots+cbor

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations section of RFC 9132.

Interoperability considerations: N/A

Published specification: RFC 9132

Applications that use this media type: DOTS agents sending DOTS messages over CoAP over (D)TLS.

Fragment identifier considerations: N/A

Additional information:
Deprecated alias names for this type: N/A
Magic number(s): N/A
File extension(s): N/A
Macintosh file type code(s): N/A

Person & email address to contact for further information:
IESG, iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: none

Author: See Authors' Addresses section.

Change controller: IESG

Provisional registration? No

10.4. CoAP Content-Formats Registration

IANA has updated the "application/dots+cbor" media type in the "CoAP Content-Formats" registry [IANA-CoAP-Content-Formats] as follows:

Media Type: application/dots+cbor
Encoding: -
ID: 271
Reference: [RFC9132]

10.5. CBOR Tag Registration

This section defines the DOTS CBOR tag as another means for applications to declare that a CBOR data structure is a DOTS signal channel object. Its use is optional and is intended for use in cases in which this information would not otherwise be known. The DOTS CBOR tag is not required for the DOTS signal channel protocol version specified in this document. If present, the DOTS tag **MUST** prefix a DOTS signal channel object.

IANA has updated the DOTS signal channel CBOR tag in the "CBOR Tags" registry [IANA-CBOR-Tags] as follows:

Tag: 271
Data Item: DDoS Open Threat Signaling (DOTS) signal channel object
Semantics: DDoS Open Threat Signaling (DOTS) signal channel object, as defined in [RFC9132]
Reference: [RFC9132]

10.6. DOTS Signal Channel Protocol Registry

The following sections update the "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel" subregistries [REG-DOTS].

10.6.1. DOTS Signal Channel CBOR Key Values Subregistry

The structure of this subregistry is provided in Section 10.6.1.1.

10.6.1.1. Registration Template

IANA has updated the allocation policy of "DOTS Signal Channel CBOR Key Values" registry as follows:

Parameter name:
Parameter name, as used in the DOTS signal channel.

CBOR Key Value:
Key value for the parameter. The key value **MUST** be an integer in the 1-65535 range.

OLD:

Range	Registration Procedures	Note
1-16383	IETF Review	comprehension-required
16384-32767	Specification Required	comprehension-optional
32768-49151	IETF Review	comprehension-optional
49152-65535	Private Use	comprehension-optional

Table 7

NEW:

Range	Registration Procedures	Note
1-127	IETF Review	comprehension-required
128-255	IETF Review	comprehension-optional
256-16383	IETF Review	comprehension-required
16384-32767	Specification Required	comprehension-optional
32768-49151	IETF Review	comprehension-optional
49152-65535	Private Use	comprehension-optional

Table 8

Registration requests for the 16384-32767 range are evaluated after a three-week review period on the dots-signal-reg-review@ietf.org mailing list, on the advice of one or more designated experts. However, to allow for the allocation of values prior to publication, the designated experts may approve registration once they are satisfied that such a specification will be published. New registration requests should be sent in the form of an email to the review mailing list; the request should use an appropriate subject (e.g., "Request to register CBOR Key Value for DOTS: example"). IANA will only accept new registrations from the designated experts, and it will check that review was requested on the mailing list in accordance with these procedures.

Within the review period, the designated experts will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful. Registration requests that are undetermined for a period longer than 21 days can be brought to the IESG's attention (using the iesg@ietf.org mailing list) for resolution.

Criteria that should be applied by the designated experts include determining whether the proposed registration duplicates existing functionality, whether it is likely to be of general applicability or whether it is useful only for a single use case, and whether the registration description is clear. IANA must only accept registry updates to the 16384-32767 range from the designated experts and should direct all requests for registration to the review mailing list. It is suggested that multiple designated experts be appointed. In cases where a registration decision could be perceived as creating a conflict of interest for a particular expert, that expert should defer to the judgment of the other experts.

CBOR Major Type:
CBOR Major type and optional tag for the parameter.

Change Controller:
For Standards Track RFCs, list the "IESG". For others, give the name of the responsible party. Other details (e.g., email address) may also be included.

Specification Document(s):
Reference to the document or documents that specify the parameter, preferably including URIs that can be used to retrieve copies of the documents. An indication of the relevant sections may also be included but is not required.

10.6.1.2. Update Subregistry Content

IANA has updated entries in the "0-51" and "49152-65535" ranges from the "DOTS Signal Channel CBOR Key Values" registry to refer this RFC.

10.6.2. Status Codes Subregistry

IANA has updated the following entries from the "DOTS Signal Channel Status Codes" registry to refer to this RFC:

Code	Label	Description	Reference
0	Reserved		[RFC9132]
1	attack-mitigation-in-progress	Attack mitigation setup is in progress (e.g., changing the network path to redirect the inbound traffic to a DOTS mitigator).	[RFC9132]
2	attack-successfully-mitigated	Attack is being successfully mitigated (e.g., traffic is redirected to a DDoS mitigator and attack traffic is dropped).	[RFC9132]
3	attack-stopped	Attack has stopped and the DOTS client can withdraw the mitigation request.	[RFC9132]
4	attack-exceeded-capability	Attack has exceeded the mitigation provider capability.	[RFC9132]

Code	Label	Description	Reference
5	dots-client-withdrawn-mitigation	DOTS client has withdrawn the mitigation request and the mitigation is active but terminating.	[RFC9132]
6	attack-mitigation-terminated	Attack mitigation is now terminated.	[RFC9132]
7	attack-mitigation-withdrawn	Attack mitigation is withdrawn.	[RFC9132]
8	attack-mitigation-signal-loss	Attack mitigation will be triggered for the mitigation request only when the DOTS signal channel session is lost.	[RFC9132]
9-2147483647	Unassigned		

Table 9: Initial DOTS Signal Channel Status Codes

New codes can be assigned via Standards Action [RFC8126].

10.6.3. Conflict Status Codes Subregistry

IANA has updated the following entries from the "DOTS Signal Channel Conflict Status Codes" registry to refer to this RFC.

Code	Label	Description	Reference
0	Reserved		[RFC9132]
1	request-inactive-other-active	DOTS server has detected conflicting mitigation requests from different DOTS clients. This mitigation request is currently inactive until the conflicts are resolved. Another mitigation request is active.	[RFC9132]
2	request-active	DOTS server has detected conflicting mitigation requests from different DOTS clients. This mitigation request is currently active.	[RFC9132]
3	all-requests-inactive	DOTS server has detected conflicting mitigation requests from different DOTS clients. All conflicting mitigation requests are inactive.	[RFC9132]

Code	Label	Description	Reference
4-2147483647	Unassigned		

Table 10: Initial DOTS Signal Channel Conflict Status Codes

New codes can be assigned via Standards Action [RFC8126].

10.6.4. Conflict Cause Codes Subregistry

IANA has updated the following entries from the "DOTS Signal Channel Conflict Cause Codes" registry to refer to this document:

Code	Label	Description	Reference
0	Reserved		[RFC9132]
1	overlapping-targets	Overlapping targets.	[RFC9132]
2	conflict-with-acceptlist	Conflicts with an existing accept-list. This code is returned when the DDoS mitigation detects source addresses/prefixes in the accept-listed ACLs are attacking the target.	[RFC9132]
3	cuid-collision	CUID Collision. This code is returned when a DOTS client uses a 'cuid' that is already used by another DOTS client.	[RFC9132]
4-2147483647	Unassigned		

Table 11: Initial DOTS Signal Channel Conflict Cause Codes

New codes can be assigned via Standards Action [RFC8126].

10.6.5. Attack Status Codes Subregistry

IANA has updated the following entries from the "DOTS Signal Channel Attack Status Codes" registry to refer to this RFC:

Code	Label	Description	Reference
0	Reserved		[RFC9132]
1	under-attack	The DOTS client determines that it is still under attack.	[RFC9132]
2	attack-successfully-mitigated	The DOTS client determines that the attack is successfully mitigated.	[RFC9132]

Code	Label	Description	Reference
3-2147483647	Unassigned		

Table 12: Initial DOTS Signal Channel Attack Status Codes

New codes can be assigned via Standards Action [RFC8126].

10.7. DOTS Signal Channel YANG Modules

IANA has registered the following URIs in the "ns" subregistry within the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-dots-signal-channel
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:iana-dots-signal-channel
Registrant Contact: IANA.
XML: N/A; the requested URI is an XML namespace.

IANA has updated the following YANG module in the "YANG Module Names" subregistry [RFC6020] within the "YANG Parameters" registry.

Name: iana-dots-signal-channel
Maintained by IANA: Y
Namespace: urn:ietf:params:xml:ns:yang:iana-dots-signal-channel
Prefix: iana-dots-signal
Reference: [RFC9132]

IANA has registered the additional following YANG module in the "YANG Module Names" subregistry [RFC6020] within the "YANG Parameters" registry. This obsoletes the registration in [RFC8782].

Name: ietf-dots-signal-channel
Maintained by IANA: N
Namespace: urn:ietf:params:xml:ns:yang:ietf-dots-signal-channel
Prefix: dots-signal
Reference: [RFC9132]

This document obsoletes the initial version of the IANA-maintained iana-dots-signal-channel YANG module (Section 5.2 of [RFC8782]). IANA is requested to maintain this note:

> Status, conflict status, conflict cause, and attack status values must not be directly added to the iana-dots-signal-channel YANG module. They must instead be respectively added to the "DOTS Status Codes", "DOTS Conflict Status Codes", "DOTS Conflict Cause Codes", and "DOTS Attack Status Codes" registries.

When a 'status', 'conflict-status', 'conflict-cause', or 'attack-status' value is respectively added to the "DOTS Status Codes", "DOTS Conflict Status Codes", "DOTS Conflict Cause Codes", or "DOTS Attack Status Codes" registry, a new "enum" statement must be added to the iana-dots-signal-channel YANG module. The following "enum" statement, and substatements thereof, should be defined:

"enum": Replicates the label from the registry.

"value": Contains the IANA-assigned value corresponding to the 'status', 'conflict-status', 'conflict-cause', or 'attack-status'.

"description": Replicates the description from the registry.

"reference": Replicates the reference from the registry and adds the title of the document.

When the iana-dots-signal-channel YANG module is updated, a new "revision" statement must be added in front of the existing revision statements.

IANA has updated this note in "DOTS Status Codes", "DOTS Conflict Status Codes", "DOTS Conflict Cause Codes", and "DOTS Attack Status Codes" registries:

> When this registry is modified, the YANG module iana-dots-signal-channel must be updated as defined in [RFC9132].

11. Security Considerations

High-level DOTS security considerations are documented in [RFC8612] and [RFC8811].

Authenticated encryption **MUST** be used for data confidentiality and message integrity. The interaction between the DOTS agents requires Datagram Transport Layer Security (DTLS) or Transport Layer Security (TLS) with a cipher suite offering confidentiality protection, and the guidance given in [RFC7525] **MUST** be followed to avoid attacks on (D)TLS. The (D)TLS protocol profile used for the DOTS signal channel is specified in Section 7.

If TCP is used between DOTS agents, an attacker may be able to inject RST packets, bogus application segments, etc., regardless of whether TLS authentication is used. Because the application data is TLS protected, this will not result in the application receiving bogus data, but it will constitute a DoS on the connection. This attack can be countered by using TCP Authentication Option (TCP-AO) [RFC5925]. Although not widely adopted, if TCP-AO is used, then any bogus packets injected by an attacker will be rejected by the TCP-AO integrity check and therefore will never reach the TLS layer.

If the 'cuid' is guessable, a misbehaving DOTS client from within the client's domain can use the 'cuid' of another DOTS client of the domain to delete or alter active mitigations. For this attack to succeed, the misbehaving client's messages need to pass the security validation checks by the DOTS server and, if the communication involves a client-domain DOTS gateway, the security checks of that gateway.

A similar attack can be achieved by a compromised DOTS client that can sniff the TLS 1.2 handshake: use the client certificate to identify the 'cuid' used by another DOTS client. This attack is not possible if algorithms such as version 4 Universally Unique IDentifiers (UUIDs) in Section 4.4 of [RFC4122] are used to generate the 'cuid' because such UUIDs are not a deterministic function of the client certificate. Likewise, this attack is not possible with TLS 1.3 because most of the TLS handshake is encrypted and the client certificate is not visible to eavesdroppers.

A compromised DOTS client can collude with a DDoS attacker to send a mitigation request for a target resource, get the mitigation efficacy from the DOTS server, and convey the mitigation efficacy to the DDoS attacker to possibly change the DDoS attack strategy. Obviously, signaling an attack by the compromised DOTS client to the DOTS server will trigger attack mitigation. This attack can be prevented by monitoring and auditing DOTS clients to detect misbehavior and to deter misuse and by only authorizing the DOTS client to request mitigation for specific target resources (e.g., an application server is authorized to request mitigation for its IP addresses, but a DDoS mitigator can request mitigation for any target resource in the network). Furthermore, DOTS clients are typically co-located on network security services (e.g., firewall), and a compromised security service potentially can do a lot more damage to the network.

Rate-limiting DOTS requests, including those with new 'cuid' values, from the same DOTS client defend against DoS attacks that would result in varying the 'cuid' to exhaust DOTS server resources. Rate-limit policies **SHOULD** be enforced on DOTS gateways (if deployed) and DOTS servers.

In order to prevent leaking internal information outside a client's domain, DOTS gateways located in the client domain **SHOULD NOT** reveal the identification information that pertains to internal DOTS clients (e.g., source IP address, client's hostname) unless explicitly configured to do so.

DOTS servers **MUST** verify that requesting DOTS clients are entitled to trigger actions on a given IP prefix. A DOTS server **MUST NOT** authorize actions due to a DOTS client request unless those actions are limited to that DOTS client's domain IP resources. The exact mechanism for the DOTS servers to validate that the target prefixes are within the scope of the DOTS client domain is deployment specific.

The presence of DOTS gateways may lead to infinite forwarding loops, which is undesirable. To prevent and detect such loops, this document uses the Hop-Limit Option.

When FQDNs are used as targets, the DOTS server **MUST** rely upon DNS privacy-enabling protocols (e.g., DNS over TLS [RFC7858] or DNS over HTTPS (DoH) [RFC8484]) to prevent eavesdroppers from possibly identifying the target resources protected by the DDoS mitigation service to ensure the target FQDN resolution is authentic (e.g., DNSSEC [RFC4034]).

CoAP-specific security considerations are discussed in Section 11 of [RFC7252], while CBOR-related security considerations are discussed in Section 10 of [RFC8949].

This document defines YANG data structures that are meant to be used as an abstract representation of DOTS signal channel messages. As such, the "ietf-dots-signal-channel" module does not introduce any new vulnerabilities beyond those specified above.

12. References

12.1. Normative References

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[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>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[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>.

[RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <https://www.rfc-editor.org/info/rfc4279>.

[RFC4632] Fuller, V. and T. Li, "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan", BCP 122, RFC 4632, DOI 10.17487/RFC4632, August 2006, <https://www.rfc-editor.org/info/rfc4632>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

[RFC7918] Langley, A., Modadugu, N., and B. Moeller, "Transport Layer Security (TLS) False Start", RFC 7918, DOI 10.17487/RFC7918, August 2016, <https://www.rfc-editor.org/info/rfc7918>.

[RFC7924] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", RFC 7924, DOI 10.17487/RFC7924, July 2016, <https://www.rfc-editor.org/info/rfc7924>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/info/rfc7959>.

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/info/rfc8085>.

[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>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/info/rfc8305>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/info/rfc8323>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

[RFC8768] Boucadair, M., Reddy.K, T., and J. Shallow, "Constrained Application Protocol (CoAP) Hop-Limit Option", RFC 8768, DOI 10.17487/RFC8768, March 2020, <https://www.rfc-editor.org/info/rfc8768>.

[RFC8783] Boucadair, M., Ed. and T. Reddy.K, Ed., "Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel Specification", RFC 8783, DOI 10.17487/RFC8783, May 2020, <https://www.rfc-editor.org/info/rfc8783>.

[RFC8791] Bierman, A., Björklund, M., and K. Watsen, "YANG Data Structure Extensions", RFC 8791, DOI 10.17487/RFC8791, June 2020, <https://www.rfc-editor.org/info/rfc8791>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

12.2. Informative References

[CORE-COMI] Veillette, M., Ed., Stok, P., Ed., Pelov, A., Bierman, A., and I. Petrov, "CoAP Management Interface (CORECONF)", Work in Progress, Internet-Draft, draft-ietf-core-comi-11, 17 January 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-core-comi-11>.

[CORE-YANG-CBOR] Veillette, M., Ed., Petrov, I., Ed., and A. Pelov, "CBOR Encoding of Data Modeled with YANG", Work in Progress, Internet-Draft, draft-ietf-core-yang-cbor-16, 25 January 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-core-yang-cbor-16>.

[DOTS-EARLYDATA] Boucadair, M. and T. Reddy.K, "Using Early Data in DOTS", Work in Progress, Internet-Draft, draft-boucadair-dots-earlydata-00, 29 January 2019, <https://datatracker.ietf.org/doc/html/draft-boucadair-dots-earlydata-00>.

[DOTS-MULTIHOMING] Boucadair, M., Reddy.K, T., and W. Pan, "Multi-homing Deployment Considerations for Distributed-Denial-of-Service Open Threat Signaling (DOTS)", Work in Progress, Internet-Draft, draft-ietf-dots-multihoming-07, 6 July 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-dots-multihoming-07>.

[DOTS-TELEMETRY] Boucadair, M., Ed., Reddy.K, T., Ed., Doron, E., Chen, M., and J. Shallow, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Telemetry", Work in Progress, Internet-Draft, draft-ietf-dots-telemetry-16, 8 December 2020, <https://datatracker.ietf.org/doc/html/draft-ietf-dots-telemetry-16>.

[IANA-CBOR-Tags] IANA, "Concise Binary Object Representation (CBOR) Tags", <https://www.iana.org/assignments/cbor-tags>.

[IANA-CoAP-Content-Formats] IANA, "CoAP Content-Formats", <https://www.iana.org/assignments/core-parameters>.

[IANA-MediaTypes] IANA, "Media Types", <https://www.iana.org/assignments/media-types>.

[IANA-Proto] IANA, "Protocol Numbers", <https://www.iana.org/assignments/protocol-numbers>.

[REG-DOTS] IANA, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel", <https://www.iana.org/assignments/dots>.

[RFC3022] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, DOI 10.17487/RFC3022, January 2001, <https://www.rfc-editor.org/info/rfc3022>.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <https://www.rfc-editor.org/info/rfc4034>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <https://www.rfc-editor.org/info/rfc4340>.

[RFC4732] Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, DOI 10.17487/RFC4732, December 2006, <https://www.rfc-editor.org/info/rfc4732>.

[RFC4787] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <https://www.rfc-editor.org/info/rfc4787>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <https://www.rfc-editor.org/info/rfc4987>.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <https://www.rfc-editor.org/info/rfc5925>.

[RFC6052] Bao, C., Huitema, C., Bagnulo, M., Boucadair, M., and X. Li, "IPv6 Addressing of IPv4/IPv6 Translators", RFC 6052, DOI 10.17487/RFC6052, October 2010, <https://www.rfc-editor.org/info/rfc6052>.

[RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <https://www.rfc-editor.org/info/rfc6146>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC6296] Wasserman, M. and F. Baker, "IPv6-to-IPv6 Network Prefix Translation", RFC 6296, DOI 10.17487/RFC6296, June 2011, <https://www.rfc-editor.org/info/rfc6296>.

[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <https://www.rfc-editor.org/info/rfc6724>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC6887] Wing, D., Ed., Cheshire, S., Boucadair, M., Penno, R., and P. Selkirk, "Port Control Protocol (PCP)", RFC 6887, DOI 10.17487/RFC6887, April 2013, <https://www.rfc-editor.org/info/rfc6887>.

[RFC6888] Perreault, S., Ed., Yamagata, I., Miyakawa, S., Nakagawa, A., and H. Ashida, "Common Requirements for Carrier-Grade NATs (CGNs)", BCP 127, RFC 6888, DOI 10.17487/RFC6888, April 2013, <https://www.rfc-editor.org/info/rfc6888>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/info/rfc7030>.

[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>.

[RFC7452] Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson, "Architectural Considerations in Smart Object Networking", RFC 7452, DOI 10.17487/RFC7452, March 2015, <https://www.rfc-editor.org/info/rfc7452>.

[RFC7589] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication", RFC 7589, DOI 10.17487/RFC7589, June 2015, <https://www.rfc-editor.org/info/rfc7589>.

[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>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[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>.

[RFC8489] Petit-Huguenin, M., Salgueiro, G., Rosenberg, J., Wing, D., Mahy, R., and P. Matthews, "Session Traversal Utilities for NAT (STUN)", RFC 8489, DOI 10.17487/RFC8489, February 2020, <https://www.rfc-editor.org/info/rfc8489>.

[RFC8499] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", BCP 219, RFC 8499, DOI 10.17487/RFC8499, January 2019, <https://www.rfc-editor.org/info/rfc8499>.

[RFC8612] Mortensen, A., Reddy, T., and R. Moskowitz, "DDoS Open Threat Signaling (DOTS) Requirements", RFC 8612, DOI 10.17487/RFC8612, May 2019, <https://www.rfc-editor.org/info/rfc8612>.

[RFC8782] Reddy.K, T., Ed., Boucadair, M., Ed., Patil, P., Mortensen, A., and N. Teague, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification", RFC 8782, DOI 10.17487/RFC8782, May 2020, <https://www.rfc-editor.org/info/rfc8782>.

[RFC8811] Mortensen, A., Ed., Reddy.K, T., Ed., Andreasen, F., Teague, N., and R. Compton, "DDoS Open Threat Signaling (DOTS) Architecture", RFC 8811, DOI 10.17487/RFC8811, August 2020, <https://www.rfc-editor.org/info/rfc8811>.

[RFC8903] Dobbins, R., Migault, D., Moskowitz, R., Teague, N., Xia, L., and K. Nishizuka, "Use Cases for DDoS Open Threat Signaling", RFC 8903, DOI 10.17487/RFC8903, May 2021, <https://www.rfc-editor.org/info/rfc8903>.

[RFC8973] Boucadair, M. and T. Reddy.K, "DDoS Open Threat Signaling (DOTS) Agent Discovery", RFC 8973, DOI 10.17487/RFC8973, January 2021, <https://www.rfc-editor.org/info/rfc8973>.

[TLS-DTLS13] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-dtls13-43>.

[URI] IANA, "Well-Known URIs", <https://www.iana.org/assignments/well-known-uris>.

Appendix A. Summary of Changes From RFC 8782

The main changes compared to [RFC8782] are as follows:

- Update the "ietf-dots-signal-channel" YANG module (Section 5.3) and the tree structure (Section 5.1) to follow the new YANG data structure specified in [RFC8791]. In particular:
  - Add in 'choice' to indicate the communication direction in which a data node applies. If no 'choice' is indicated, a data node can appear in both directions (i.e., from DOTS clients to DOTS servers and vice versa).
  - Remove 'config' clauses. Note that 'config' statements will be ignored (if present) anyway, according to Section 4 of [RFC8791]. This supersedes the references to the use of 'ro' and 'rw', which are now covered by 'choice' above.
  - Remove 'cuid', 'cdid', and 'sid' data nodes from the structure because these data nodes are included as Uri-Path options, not within the message body.
  - Remove the list keys for the mitigation scope message type (i.e., 'cuid' and 'mid'). 'mid' is not indicated as a key because it is included as a Uri-Path option for requests and in the message body for responses. Note that Section 4 of [RFC8791] specifies that a list does not require to have a key statement defined.
- Add a new section with a summary of the error code responses that can be returned by a DOTS server (Section 9).
- Update the IANA section to allocate a new range for comprehension-optional attributes (Section 10.6.1.1). This modification is motivated by the need to allow for compact DOTS

signal messages that include a long list of comprehension-optional attributes, e.g., DOTS telemetry messages [DOTS-TELEMETRY].

- Add Appendix C to list recommended/default values of key DOTS signal channel parameters.
- Add subsections to Section 4.4.1 for better readability.

Appendix B. CUID Generation

The document recommends the use of SPKI to generate the 'cuid'. This design choice is motivated by the following reasons:

- SPKI is globally unique.
- It is deterministic.
- It allows the avoidance of extra cycles that may be induced by 'cuid' collision.
- DOTS clients do not need to store the 'cuid' in a persistent storage.
- It allows the detection of compromised DOTS clients that do not adhere to the 'cuid' generation algorithm.

Appendix C. Summary of Protocol Recommended/Default Values

Parameter	Recommended/Default Value
Port number	4646 (tcp/udp)
lifetime	3600 seconds
active-but-terminating	120 seconds
maximum active-but-terminating	300 seconds
heartbeat-interval	30 seconds
minimum 'heartbeat-interval'	15 seconds
maximum 'heartbeat-interval'	240 seconds
missing-hb-allowed	15
max-retransmit	3
ack-timeout	2 seconds
ack-random-factor	1.5
probing-rate	5 bytes/second

Parameter	Recommended/Default Value
trigger-mitigation	true

Table 13

Acknowledgements

Many thanks to Martin Björklund for the suggestion to use [RFC8791].

Thanks to Valery Smyslov for the comments, guidance, and support.

Thanks to Ebben Aries for the yangdoctors review, Dan Romascanu for the opsdir review, Michael Tuexen for the tsv-art review, Dale Worley for the genart review, and Donald Eastlake 3rd for the secdir review.

Thanks to Benjamin Kaduk for the AD review.

Thanks to Martin Duke, Lars Eggert, Erik Kline, Murray Kucherawy, Éric Vyncke, and Robert Wilton for the IESG review.

Acknowledgements from RFC 8782

Thanks to Christian Jacquenet, Roland Dobbins, Roman Danyliw, Michael Richardson, Ehud Doron, Kaname Nishizuka, Dave Dolson, Liang Xia, Gilbert Clark, Xialiang Frank, Jim Schaad, Klaus Hartke, Nesredien Suleiman, Stephen Farrell, and Yoshifumi Nishida for the discussion and comments.

The authors would like to give special thanks to Kaname Nishizuka and Jon Shallow for their efforts in implementing the protocol and performing interop testing at IETF Hackathons.

Thanks to the core WG for the recommendations on Hop-Limit and redirect signaling.

Special thanks to Benjamin Kaduk for the detailed AD review.

Thanks to Alexey Melnikov, Adam Roach, Suresh Krishnan, Mirja Kuehlewind, and Alissa Cooper for the review.

Thanks to Carsten Bormann for his review of the DOTS heartbeat mechanism.

Contributors

The authors of RFC 8782 are listed below:

Tirumaleswar Reddy.K (editor)
McAfee, Inc.
Embassy Golf Link Business Park
Bangalore 560071
Karnataka
India
Email: kondtir@gmail.com

Mohamed Boucadair (editor)
Orange
35000 Rennes
France
Email: mohamed.boucadair@orange.com

Prashanth Patil
Cisco Systems, Inc.
Email: praspati@cisco.com

Andrew Mortensen
Arbor Networks, Inc.
2727 S. State Street
Ann Arbor, MI 48104
United States of America
Email: andrew@moretension.com

Nik Teague
Iron Mountain Data Centers
United Kingdom
Email: nteague@ironmountain.co.uk

The following individuals have contributed to RFC 8782:

Jon Shallow
NCC Group
Email: jon.shallow@nccgroup.trust

Mike Geller
Cisco Systems, Inc.
FL 33309
United States of America
Email: mgeller@cisco.com

Robert Moskowitz
HTT Consulting
Oak Park, MI 42837
United States of America
Email: rgm@htt-consult.com

Authors' Addresses

Mohamed Boucadair (EDITOR)
Orange
35000 Rennes
France
Email: mohamed.boucadair@orange.com

Jon Shallow
United Kingdom
Email: supjps-ietf@jpshallow.com

Tirumaleswar Reddy.K
Akamai
Embassy Golf Link Business Park
Bangalore 560071
Karnataka
India
Email: kondtir@gmail.com