DOTS
Internet-Draft
Intended status: Standards Track
Expires: August 19, 2017

N. Teague
Verisign, Inc.
A. Mortensen
Arbor Networks, Inc.
February 15, 2017

DDoS Open Threat Signaling Protocol
draft-teague-dots-protocol-02

Abstract

This document describes Distributed-Denial-of-Service (DDoS) Open Threat Signaling (DOTS), a protocol for requesting and managing mitigation of DDoS attacks.

DOTS mitigation requests over the signal channel permit domains to signal the need for help fending off DDoS attacks, setting the scope and duration of the requested mitigation. Elements called DOTS servers field the signals for help, and enable defensive countermeasures to defend against the attack reported by the clients, reporting the status of the delegated defense to the requesting clients. DOTS clients additionally may use a reliable data channel to manage filters and black- and white-lists to restrict or allow traffic to the clients' domains arbitrarily. The DOTS signal channel may operate over UDP [RFC0768] and if necessary TCP [RFC0793]. This revision discusses a transport-agnostic approach to this channel, focusing on the message exchanges and delegating transport specifics to other documents. Discussion of the reliable data channel may be found in [I-D.reddy-dots-data-channel].

Status of This Memo

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

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

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

This Internet-Draft will expire on August 19, 2017.

Table of Contents

1. Introduction

Distributed-Denial-of-Service attack scale and frequency continues to increase year over year, and the trend shows no signs of abating [WISR]. In response to the DDoS attack trends, service providers and vendors have developed various approaches to sharing or delegating responsibility for defense, among them ad hoc service relationships, filtering through peering relationships [COMMUNITYFS], and proprietary solutions ([CLOUDSIGNAL], [OPENHYBRID]). Such hybrid approaches to DDoS defense have proven effective, but the heterogeneous methods employed to coordinate DDoS defenses across domain boundaries have necessarily limited their scope and effectiveness, as the mechanisms in one domain have no traction in another.

The DDoS Open Threat Signaling (DOTS) protocol provides a common mechanism to achieve the coordinated attack response previously restricted to custom or proprietary solutions. To meet the needs of network operators facing down modern DDoS attacks, DOTS itself is a hybrid protocol, consisting of a signal channel and a data channel. DOTS uses the signal channel, a lightweight and robust communication layer, to signal the need for mitigation regardless of network conditions, and uses the reliable data channel as vehicle for provisioning, configuration, telemetry, filter management, and other unsized or bulk data exchange.

The DOTS protocol is not intended as a replacement for such protocols as BGP Flow Specification [RFC5575] or as a general purpose DDoS countermeasure application programming interface (API), but rather as an advisory protocol enabling attack response coordination between DOTS agents. Any DOTS-enabled device or service is capable of triggering a request for help and shaping the scope and nature of

that help, with the details of the actual mitigation left to the discretion of the operators of the attack mitigators. DOTS thereby permits all participating parties to manage their own attack defenses in the manner most appropriate for their own domains.

1.1. Terminology

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

Terms used to define entity relationships, transmitted data, and methods of communication are drawn from the terminology defined in [I-D.ietf-dots-requirements].

2. Architecture

The architecture in which the DOTS protocol operates is assumed to be derived from the architectural components and concepts described in [I-D.ietf-dots-architecture].

2.1. DOTS Agents

All protocol communication is between a DOTS client and a DOTS server. The logical agent termed a DOTS gateway is in practice a DOTS server placed back-to-back with a DOTS client. As discussed in [I-D.ietf-dots-architecture], any interface enabling the back-to-back DOTS server and client to act as a DOTS gateway is implementation-specific. This protocol is therefore concerned only with managing one or more bilateral relationships between DOTS clients and the DOTS servers, a signaling mode known as Direct Signaling in the DOTS architecture. This is shown in Figure 1 below:

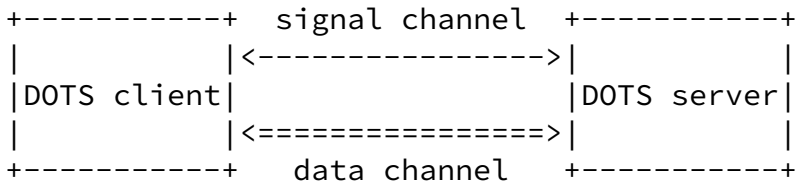


Figure 1: DOTS protocol direct signaling

The DOTS architecture anticipates many-to-one and one-to-many deployments, in which multiple DOTS clients maintain distinct signaling sessions with a single DOTS server or a single DOTS client maintains distinct signaling sessions with multiple DOTS servers, as shown below in Figure 2:

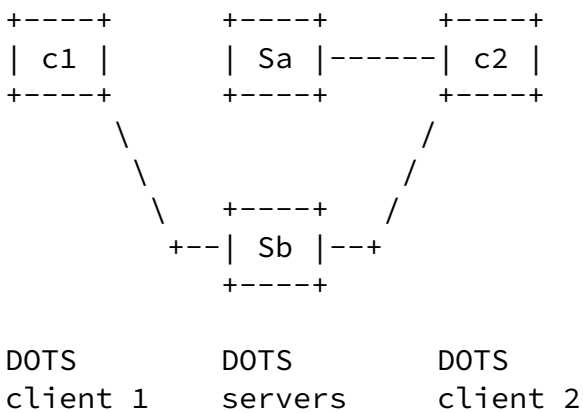


Figure 2: DOTS protocol direct signaling

DOTS server Sb has signaling sessions with DOTS clients c1 and c2. DOTS client c2 has signaling sessions with DOTS servers Sa and Sb. Except where explicitly defined in this protocol, all mechanisms to maintain multiple signaling sessions are left to the implementation.

3. DOTS Communication Channels

DOTS uses two channels, a signal channel and a data channel.

The signal channel is the minimal secure communication layer a DOTS client uses to request mitigation for resources under the administrative control the DOTS client; administrative control may be delegated.

The data channel offers DOTS clients a limited ability to manage configuration and attack filtering for requested mitigations. The data channel offers DOTS client operators the limited ability to adjust configuration and filtering for their mitigation requests.

This document describes the DOTS signal channel. The data channel is defined separately in [I-D.reddy-dots-data-channel].

4. Signal Channel

The purpose of the signaling channel is to convey DDoS mitigation request and status information between participating agents (client and server or gateway). Conditions during a DDoS attack are invariably hostile for connection-oriented protocols traversing affected paths. Mechanisms such as Happy Eyeballs [RFC6555] may be used to select a transport suitable for a given time and prevailing network conditions.

Implementations are required to support the DOTS signal channel over UDP as specified in [I-D.ietf-dots-requirements]. This document

therefore assumes the the availability of a transport based upon UDP [RFC0768], but also defines the message exchanges agnostic of the underlying transport used to convey the signaling.

Key tenets of DOTS protocol design are low communication overhead and efficient message packing to increase the chances of successful transmission and receipt. Desirable side-effects of efficient packing are the removal of the possibility of fragmentation in addition to a message size that is friendly towards encapsulation (e.g via GRE [RFC2784] or MPLS [RFC3031]). Large UDP packets may also be treated adversely by middleboxes with restrictive policies or may fall foul of aggressive filtering.

In support of operational requirements for protocol efficiency, backward compatibility and extensibility in

[I-D.ietf-dots-requirements], the signaling channel uses Protocol Buffers [PROTOBUF], also known as Protobufs, to define message schemas and serialize messages exchanged between DOTS agents.

The security requirements described in [I-D.ietf-dots-requirements] (peer mutual authentication, message confidentiality and integrity, and message replay protection are not discussed here. However, these qualities must be present in the transport over which the DOTS signal channel operates.

Key distribution in support of those security requirements to may be achieved via the data channel, via an online mechanism such as DANE [RFC6698], Enrollment over Secure Transport [RFC7030], or by out-of-band means. The key distribution mechanism is not specified in this document, but operators must take care to ensure the distribution provides the same level of security demanded by the signal channel itself.

4.1. Minimum Viable Information

DOTS is intended to be extensible and to evolve to meet the future needs in communicating as yet unknown threats. However, it must be able to convey the minimum information required for an upstream mitigation platform to successfully counter a DDoS attack. A client may have limited visibility into the full breadth of an attack and as such may not be well placed to provide useful telemetry. DDoS sources may or may not be spoofed and number in the millions. Once mitigation is active, the filtered traffic seen by the DOTS client (or elements informing the DOTS client operator's decision to request mitigation) may not be representative of the ongoing attack. This provides challenges for the quality and usefulness of telemetry and mitigation/countermeasure stipulations and as such this type of information if conveyed can only be considered advisory.

In these instances the minimum viable information required for the majority of mitigations to be activated is that which pertains to the resource being targeted by the attack (host, prefix, protocol, port, URI etc.), per [I-D.ietf-dots-requirements] (OP-006). The DOTS requirements also identify a mitigation lifetime period (OP-005) and mitigation efficacy metric (OP-007). The former may be considered for inclusion in the minimum viable information set, however, the latter may only be relevant in updates. An explicit mitigation

request/terminate flag is also required: a mitigation MUST be explicitly requested by a DOTS client operator. Finally, each message should include a message id or sequence number field as well as a field for the last received message id or sequence number. These may then be compared by the endpoints to assist in tracking state and/or identifying loss.

4.2. Signal Channel Messages

4.2.1. Messaging Overview

The signal channel requirements in [I-D.ietf-dots-requirements] demand a protocol capable of operating despite significant link congestion between DOTS agents. In an effort to maximize the probability of signal delivery between DOTS agents, all DOTS signal channel messages in the DOTS protocol are conceptually unidirectional heartbeats sent between DOTS agents.

The result is a form of scheduled messaging between DOTS agents, in contrast to a conventional request-response model. Once a signal channel is established, a DOTS client begins sending unidirectional heartbeats to the DOTS server, separated by the configured session heartbeat interval (see Section 4.3.1 below). To request mitigation, a DOTS client merely adds the requested mitigation parameters to its heartbeat, and maintains that request until a heartbeat from the DOTS server indicates receipt of that mitigation request. The DOTS server likewise sends its unidirectional heartbeat on the schedule interval, augmenting the content of the heartbeat with mitigation request status.

A simple exchange between DOTS agents with an established signaling session is shown below in Figure 3.

Client Server

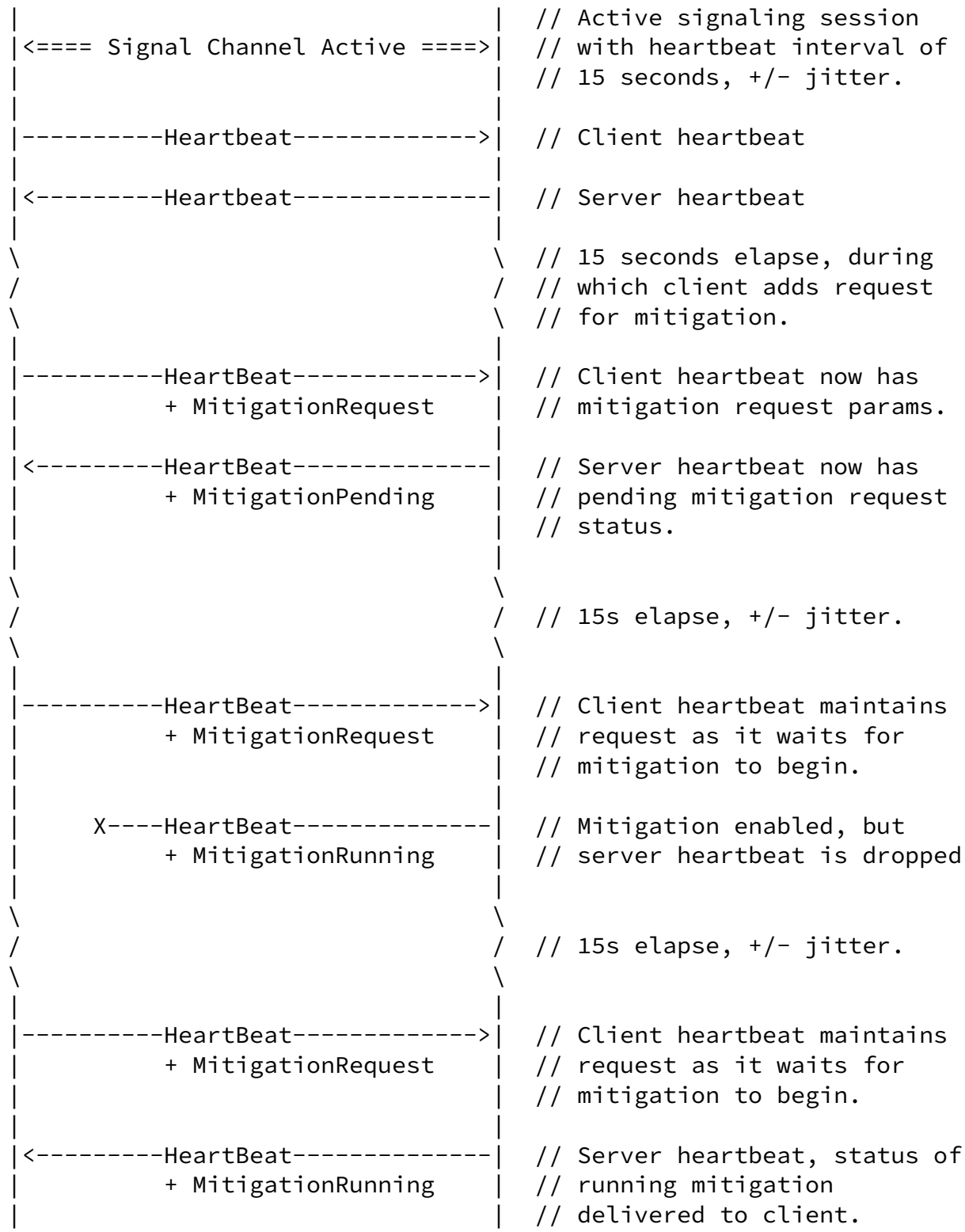


Figure 3: Signaling Session with Mitigation Request

All messages are variations of this scheduled heartbeat model. See Section 4.3 below for detailed discussion of the message exchanges between DOTS agents.

4.2.2. Message Definition and Serialization

The DOTS protocol signal channel uses Protobufs [PROTOBUF] as an interface definition language (IDL) for signal channel messaging between DOTS agent, reducing the number of discrete messages to just a single message superset per direction, with function defined by the chosen fields contained within the message.

Protobufs schemas are used to define the messages sent in either direction, from which code may be generated for specific language implementations of DOTS. As Protobufs serialization relies on numbered fields, signal channel messaging permits the introduction of new numbered fields arbitrarily, adding the requisite extensibility to the protocol while retaining backward compatibility. Future revisions of or extensions to the protocol may use the data channel to provide a mechanism by which schema updates or expansions may be communicated during provisioning/session establishment.

4.2.3. Client Message Fields

Only the seqno and last_svr_seqno fields are required in every message, as they are the minimum required for the heartbeat. Subsets of the other fields are used to convey a given signal message type.

The fields in the DOTS client signal channel message have the following functions:

seqno: a client-generated sequence number unique to the message. The client increments the seqno value by one for each message sent over the signal channel.

last_svr_seqno: the sequence number of the last message received from the server, provided to the server as a simple way to detect lost messages.

mitigations: a list of mitigations requested or withdrawn by the client. The mitigation schema fields are described below.

active: indicates a request for a list of active mitigations and their detail that are current on the DOTS server.

ping: an operator initiated heartbeat like message which will elicit a response from the DOTS server. This may be used to prove bi-directional communications on an ad-hoc basis. For example, a DOTS ping may be used to prove keying material on the DOTS client is valid and may be used to establish signaling sessions with the

DOTS server.

extensions: these fields may be used to communicate implementation specific details. An example would be the dissemination of filters between DOTS client and DOTS server.

4.2.3.1. Client Message Schema

The DOTS client message schema is detailed in Section 7.1.

4.2.4. Mitigation Request Fields

The fields in a mitigation request are as follows:

eventid: an opaque client generated identifier that distinguishes a unique event or incident. May be used by the client as a reference to the specific event triggering a mitigation request, or for other implementation-specific purposes.

requested: signals the need for mitigation to the DOTS server. If true, the DOTS client is requesting mitigation for the provided scope. If false, the DOTS client is indicating it does not require mitigation, and the DOTS server MUST cease the mitigation for the provided scope.

scope: the scope of the mitigation requested, which may be any of the types described in [I-D.ietf-dots-requirements], such as Classless Internet Domain Routing (CIDR) [RFC1518],[RFC1519] prefixes, DNS names, or aliases defined by the DOTS client operator through the data channel.

lifetime: the lifetime in seconds a mitigation request should be considered valid.

efficacy: a metric to convey to a DOTS server the perceived efficacy of an active mitigation, per operational requirements in [I-D.ietf-dots-requirements]. The mitigation efficacy is represented as a floating point value between 0 and 1, with smaller values indicating lesser efficacy, and larger greater efficacy.

extensions: these fields may be used to provide implementation-specific mitigation details.

4.2.4.1. Mitigation Request Schema

The DOTS client message schema is detailed in Section 7.2.

4.2.5. DOTS Server Message Fields

DOTS server messages use a subset of the available fields to convey the given signal type, including additional relevant fields as necessary. The only fields which may be common to all signals are seqno and last_client_seqno which may be used to detect message loss or out-of-order delivery. When conveying mitigation information, the server schema may bundle multiple mitigation status datasets into a single message, provided this does not violate the required sub-MTU message size [I-D.ietf-dots-requirements].

The fields in the DOTS server signal channel message schema have the following functions:

seqno: a server generated sequence number unique to the message.

last_cli_seqno: the seqno of the last message received from the client.

ping: an operator-initiated heartbeat like message which will elicit a response from the DOTS client. This may be used to prove bi-directional communications on an ad-hoc basis.

error: details of an error caused by a DOTS client request.

redirect: Populated with the details of the redirection target DOTS server, if the DOTS server is redirecting the DOTS client to another DOTS server.

mitigations: a list containing the status of mitigations requested by the DOTS client. The fields in the mitigation status schema are described below.

extensions: these fields may be used to communicate implementation specific details. An example would be the communication of DNS mitigation vip to the DOTS client by the DOTS server.

4.2.5.1. DOTS Server Message Schema

The server message schema is detailed in Section 7.4.

4.2.6. Server Errors

If a DOTS client message cannot be processed by the DOTS server, or for any other reason causes an error, the DOTS server MUST populate the error field in any response to the message causing the error. As the error response itself may be lost, a DOTS client may continue sending problematic messages regardless of the DOTS server's error

notifications. DOTS server implementations MAY terminate the signaling session after client-triggered errors exceed a threshold during a time period equivalent to three times the session heartbeat interval.

The DOTS client message triggering the error condition is indicated in the last_client_seqno value of the DOTS server message containing the error.

Error codes MUST be one of the following types:

NOERROR: Indicates the DOTS server has detected no error resulting from a DOTS client message. Implementations MAY omit the error field entirely when no error condition is present. This value is included in the schema largely to adhere to the convention that an error status of 0 indicates success.

INVALID_VALUE: Indicates the DOTS client included an invalid value for a field in the client message most recently received from the client. The DOTS server SHOULD include specifics of the invalid value in the details field of the error.

MITIGATION_UNAVAILABLE: Indicates the DOTS server is unable to provide mitigation in response to a mitigation request from the DOTS client.

MITIGATION_CONFLICT: Indicates a mitigation request conflicts with an existing mitigation from the client. The DOTS server SHOULD populate the error details field with the status information of the mitigation conflicting with the requested mitigation.

MALFORMED_MESSAGE: Indicates the DOTS client message is malformed and cannot be processed.

4.2.6.1. Server Error Fields

code: a numeric code categorizing the error type detected by the DOTS server.

details: specific information about the reason for the detected error.

4.2.6.2. Server Error Schema

The server error schema is detailed in Section 7.7.

4.2.7. Server Mitigation Status Fields

The DOTS server message contains zero or more mitigation status messages, the fields of which have the following functions:

eventid: an opaque client generated identifier that distinguishes a unique event or incident.

ttl: the remaining lifetime of the mitigation, in seconds.

bytes_dropped: the total dropped byte count for the mitigation associated with eventid.

bps_dropped: the dropped bytes per second for the mitigation associated with eventid. This value is expected to be calculated by the mitigator, and as such is implementation-specific.

pkts_dropped: the total dropped packet count for the mitigation

associated with eventid..

pps_dropped: the dropped packets per second for the mitigation associated with eventid. This value is expected to be calculated by the mitigator, and as such is implementation-specific.

blacklist_enabled: Indicates whether a blacklist of prohibited traffic sources is enabled for the mitigation associated with eventid. The blacklist is managed through the data channel.

whitelist_enabled: Indicates whether a whitelist of sources from which traffic must always be allowed is enabled. The whitelist is managed through the data channel.

filters_enabled: Indicates whether client-specified traffic filters are enabled for the mitigation associated with eventid.

4.3. Interactions

4.3.1. Session Initialization

Signaling sessions are initiated by the DOTS client. Session initialization begins when the DOTS client connects to the DOTS server port, 4646. After connecting, the DOTS client establishes the channel security context, including all necessary cryptographic exchanges between the two DOTS agents.

This signal channel specification is transport-agnostic, and delegates the details of transport, including transport security, to transport-specific documents. Regardless of transport, DOTS

implementations nonetheless MUST provide signal channel security meeting the requirements in [I-D.ietf-dots-requirements].

Once the signal channel security context is established, the DOTS client sends a channel initialization message to the DOTS server, optionally including signaling session configuration values; if the session configuration values are excluded, defaults MUST be used for the signaling session. An example initialization message setting the acceptable signal loss and heartbeat interval for the signaling sessions is described in Figure 4 below:

```
message DOTSClientMessage {
  1 (seqno) = %;
  2 (last_svr_seqno) = %;
  6 (config) = {
    1 (loss_limit) = %;
    3 (heartbeat_interval) = %;
  };
}
```

Figure 4: Signal Channel Initialization Message

The DOTS server MUST respond immediately by sending a heartbeat (see Section 4.3.2 below) to the DOTS client. The signal channel is active when the DOTS client receives a heartbeat from the DOTS server with a last_client_seqno of a signal channel initialization message. Both DOTS agents MUST begin sending heartbeats on the interval for the signaling session once the session is active.

The following example assumes a DOTS implementation using UDP as the transport and DTLS1.2 [RFC6347]. In Figure 5 below, the DOTS client uses the default values for acceptable signal loss, maximum mitigation lifetime, and heartbeat interval. The initial DOTS server heartbeat is lost, so the DOTS client sends another channel initialization message after waiting for the minimum heartbeat interval defined below in Section 4.3.2:


```
Client                              Server
  |                                    |
  |<- - - - DTLS1.2 handshake - - - ->|  // Server listens on UDP:4646.
  |                                    |  // Client initiates DTLS
```

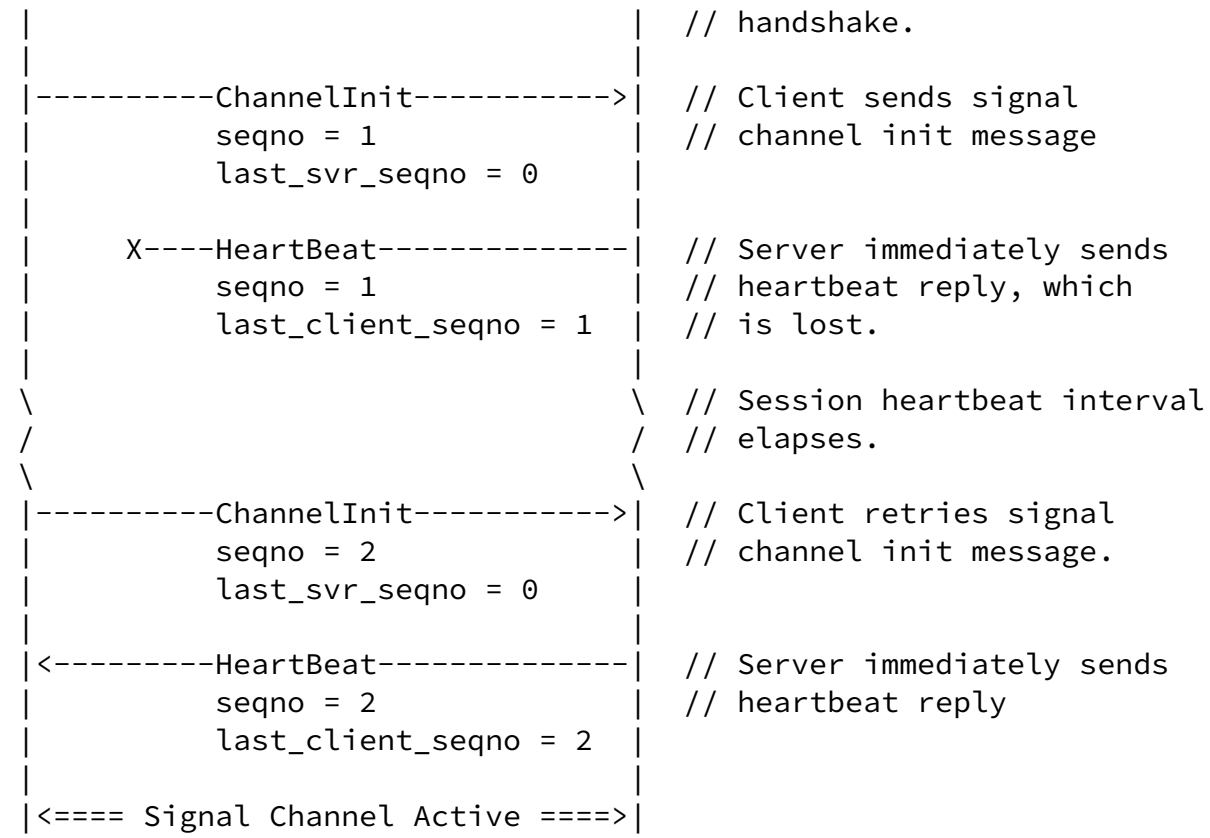


Figure 5: Signal Channel Initialization

4.3.1.1. Session Initialization Error Handling

If the DOTS client specifies invalid values for the signal channel configuration, the DOTS server replies with an error, and may ultimately terminate the connection if the client fails to correct the invalid values, as described in [I-D.ietf-dots-architecture].

4.3.1.2. Mis-Sequencing

In the event that the DOTS agent receives messages containing invalid seqno, last_client_seqno or last_svr_seqno these should be discarded and ignored.

4.3.2. Heartbeat

The most common message exchanged between a DOTS client and a DOTS server is a heartbeat (OP-002 [I-D.ietf-dots-requirements]), which maintains and monitors the health of the DOTS session. This is achieved with simple, loosely-coupled bi-directional messages

containing the sending DOTS agent's message sequence number and the sequence number the sending DOTS agent last received from its peer. Due to the stress volumetric DDoS impose upon a network, a degree of loss during attacks is to be expected. Message loss tolerance may be set on signal channel establishment.

The default heartbeat interval is 20 seconds, plus or minus a number of milliseconds between 50 and 2000. The number of milliseconds MUST be randomized in order to introduce jitter into the heartbeat interval, as recommended by [RFC5405]. The default interval is derived from the recommendations in [RFC5405] regarding middlebox traversal, to maintain NAT bindings in the path between DOTS agents.

The interval between heartbeats is may also be set by the client when establishing the signal channel. The minimum heartbeat interval is 15 seconds, plus the random number of milliseconds as described above. The maximum heartbeat interval is 120 seconds (two minutes), minus the random number of milliseconds described above.

Heartbeats are loosely-coupled, meaning each DOTS agent in a bilateral signaling session sends DOTS heartbeats on the specified interval, but asynchronously, without acknowledgement. Each DOTS agent tracks heartbeats received from its peer, and includes the sequence number of the last heartbeat received from the peer agent in the next heartbeat sent, as shown in Figure 6:

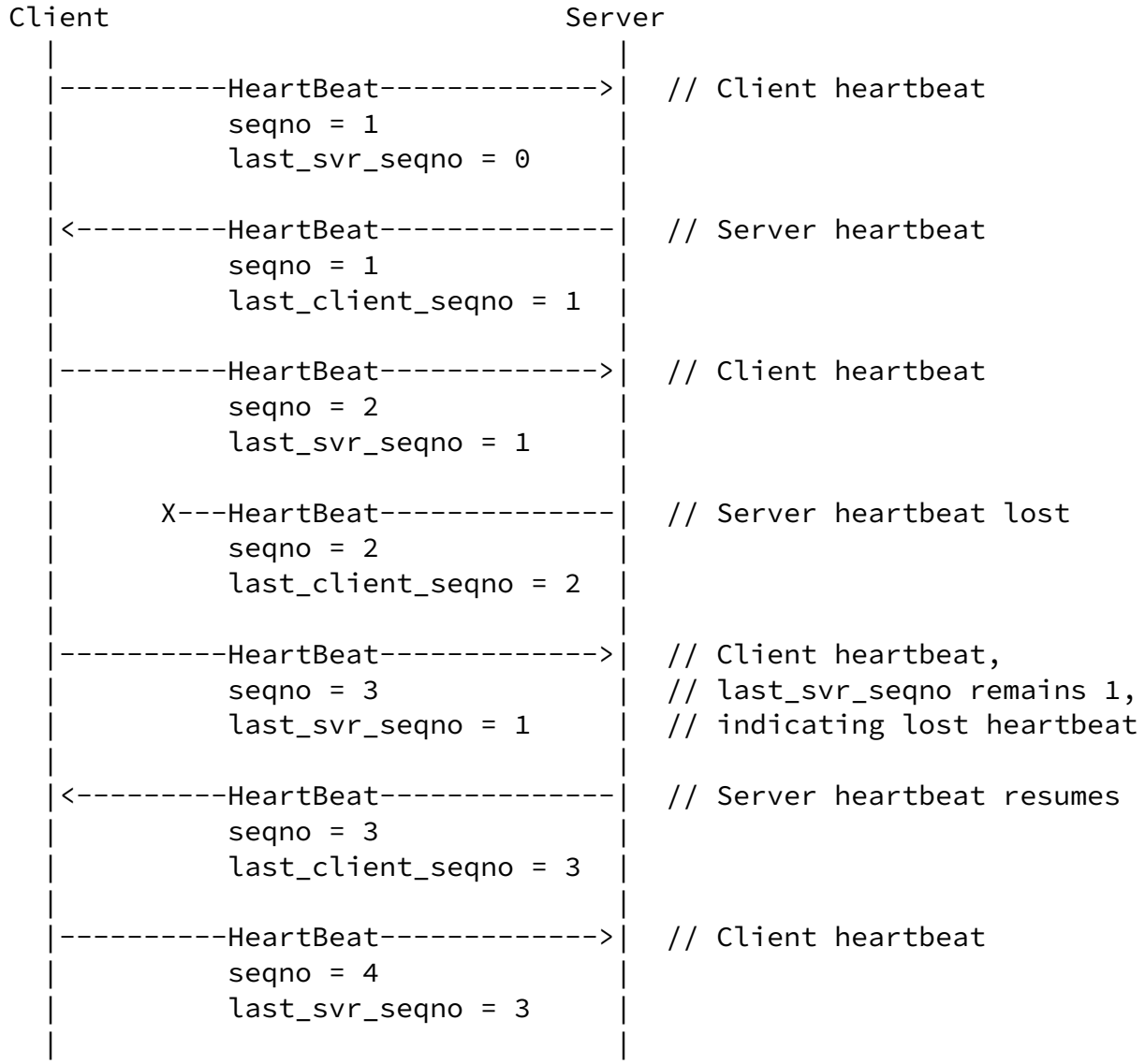


Figure 6: Heartbeats Between DOTS agents

The DOTS client heartbeat has the following format:

```
message DOTSClientMessage {
  1 (seqno) = %;
  2 (last_svr_seqno) = %;
}
```

The DOTS server heartbeat is identical aside from the schema type:

```
message DOTSServerMessage {
  1 (seqno) = %;
  2 (last_svr_seqno) = %;
}
```


Should the number of signals lost exceed the acceptable lossiness value for the signaling session, the agent detecting the signal loss may consider the signaling session lost. The default value for acceptable signal loss is 9, which, when coupled with the default heartbeat interval, amounts to lack of heartbeat from the peer DOTS agent for 180 seconds (three minutes).

4.3.2.1. Ping

There may be cases where a DOTS client or server operator wishes to trigger an immediate heartbeat response in order to validate bi-directional communication (e.g. during provisioning). This ad-hoc triggering may be achieved by setting the ping field set to TRUE. When DOTS agent receives a message on the signal channel with the ping field set to TRUE, it MUST immediately send heartbeat back to the ping sender. A ping reply MUST consist of only the senders sequence number and the sequence number of the received ping. [[EDITOR'S NOTE: rate limiting of pings required?]]

A ping is identical to a standard heartbeat, but with the ping field included and set to true:

```
message DOTSClientMessage {
  1 (seqno) = %;
  2 (last_svr_seqno) = %;
  5 (ping) = true;
}
```

4.3.3. Mitigation Request Handling

The mitigation request is the crux of the DOTS protocol, and is comprised of the minimum viable information described in Section 4.1. The request may be augmented with additional implementation specific

extensions where these do not result in undue packet bloat. The DOTS client may send repeated requests until it receives a suitable response from the DOTS server by which it may interpret successful receipt.


```
message DOTSClientMessage {
  1 (seqno) = %;
  2 (last_svr_seqno) = %;
  3 (mitigations) = [
    {
      1 (eventid) = %;
      2 (requested) = %;
      3 (scope) = %;
      4 (lifetime) = %;
    }
  ];
}
```

The DOTS server is expected to respond to confirm that it has accepted and or rejected the mitigation request. Upon receipt of the response the DOTS client should cease sending additional initial requests for the same eventid. If these do not cease then the server may assume that the response was possibly lost and should resend accordingly. Acceptance status is communicated by the DOTS server replying with the corresponding eventid and the enabled field set to 1 for acceptance and 0 for rejection. A rejection by the DOTS server should be accompanied with an extension field detailing succinctly the reason (e.g. out of contract, conflict, maintenance etc.).

```
message DOTSServerMessage {
  1 (seqno) = %;
```

```
      2 (last_cli_seqno) = %;
      4 (mitigations) = [
        {
          1 (eventid) = %;
          2 (enabled) = true; // Mitigation request accepted
        }
      ]
    }
```

After a period of time the mitigation request may expire and the DOTS server may end the mitigation. Alternately, the DOTS client may explicitly terminate the active mitigation by sending a message to the server that contains a mitigation value with the eventid and that has the requested field set to false, as shown below:

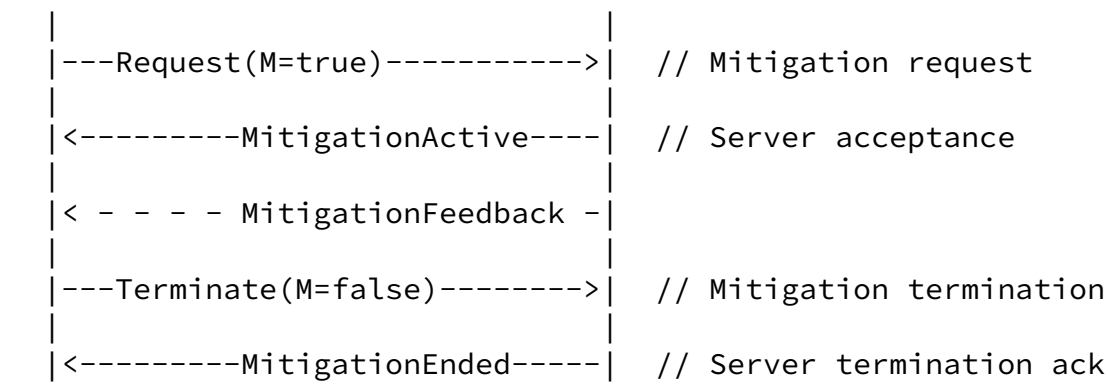

```
    message DOTSClientMessage {
      1 (seqno) = %;
      2 (last_svr_seqno) = %;
      3 (mitigations) = [
        {
          1 (eventid) = %;
          2 (requested) = false; // Terminate mitigation
        }
      ];
    }
```

The server must explicitly acknowledge the termination with a response message with the enabled field now set to false:

```
    message DOTSServerMessage {
      1 (seqno) = %;
      2 (last_cli_seqno) = %;
      6 (mitigations) = [
        {
```

```
            1 (eventid) = %;
            2 (enabled) = false; // Mitigation terminated
          }
        ];
      }
```

The life cycle of a DOTS mitigation request resembles the following:



4.3.4. Ancillary Messages

In addition to the basic interaction, additional messages may be exchanged throughout the lifetime of the mitigation. The following message types are defined to provide requisite information between DOTS agents during an active signaling session.

4.3.4.1. Mitigation Feedback

The DOTS server MUST update the client with current mitigation status. This MUST include the eventid, and SHOULD include available dropped attack traffic statistics provided by the mitigator. A DOTS server MAY provide feedback for more than one mitigation in a single message, provided the resulting message meets the sub-MTU size requirements in [I-D.ietf-dots-requirements].

The DOTS client SHOULD use the feedback from the DOTS server when deciding to update or terminate a mitigation request. For example, if the DOTS client learns from DOTS server mitigation feedback that

the dropped_pps rate is low, the DOTS client might decide to terminate upstream mitigation and handle the attack locally.

A mitigation feedback message from the DOTS server would resemble the following format, assuming an active mitigation request from the DOTS client:

```
message DOTSServerMessage {
  1 (seqno) = %;
  2 (last_client_seqno) = %;
  6 (mitigations) = [
    {
      1 (eventid) = %;
      2 (enabled) = %;
      3 (ttl) = %;
      4 (bytes_dropped) = %;
      5 (bps_dropped) = %;
      6 (pkts_dropped) = %;
      7 (pps_dropped) = %;
      10 (filters_enabled) = true;
    },
  ];
}
```

4.3.4.2. Mitigation Lifetime Update

The DOTS client may wish to update the mitigation during its lifetime. Updates may be to alter the lifetime to extend the mitigation, or an update may communicate the perceived efficacy of the mitigation. The former may be as a result of the DOTS sever feedback which may suggest that an attack shows no sign of abating. The latter may be to notify the DOTS server whether the countermeasures deployed are perceived as effective or not.

A DOTS client may update the lifetime of multiple mitigations in a single request as long as the message size meets the sub-MTU

requirement per [I-D.ietf-dots-requirements]. The lifetime update message has the following format:

```
message DOTSClientMessage {
  1 (seqno) = %;
```

```
      2 (last_svr_seqno) = %;
      3 (mitigations) = [
        {
          1 (eventid) = %;
          2 (requested) = true;
          4 (lifetime) = %;
        }
      ];
    }
```

Upon receipt of the mitigation lifetime update, the DOTS server replace the current mitigation expiration time with the new value. The updated lifetime MUST be visible in the ttl field in subsequent mitigation feedback messages. When updating a mitigation lifetime, the DOTS client SHOULD continue sending the lifetime update request at the heartbeat interval until the DOTS server's mitigation feedback shows an updated ttl for the updated mitigation.

4.3.4.3. Mitigation Efficacy Updates

When a mitigation is active, a DOTS client MUST periodically communicate the locally perceived efficacy of the mitigation to the DOTS server. This gives the DOTS server a rough sense of whether the DOTS client perceives the mitigator's deployed countermeasures as effective. The efficacy update update message has the following format:

```
message DOTSClientMessage {
  1 (seqno) = %;
  2 (last_svr_seqno) = %;
  3 (mitigations) = [
    {
      1 (eventid) = %;
      6 (efficacy) = %;
    }
  ];
}
```

The DOTS server SHOULD consider the efficacy update an indication of the effectiveness of any ongoing mitigations related to the eventid provided by the DOTS client. The DOTS server nonetheless MAY treat any efficacy update from the client as advisory, and is under no obligation to alter the mitigation strategy in response.

5. IANA Considerations

The DOTS protocol requires a well-known port to which DOTS client messages are sent. The DOTS server listens on the well-known port for client messages. The DOTS client binds to an ephemeral port per Section 4.3.1 above. This document selects port 4646 (the ASCII decimal values for "..": "DOTS") as the well-known port.

6. Security Considerations

The DOTS protocol controls mitigation request and withdrawal and as such care must be taken to protect against concerns outlined in the security considerations of [I-D.ietf-dots-architecture].

7. Appendix A: Message Schemas

7.1. DOTS Client Message Schema

```
syntax = "proto3";
import "google/protobuf/any.proto";

message DOTSClientMessage {
  // Client generated sequence number
  uint64 seqno = 1;

  // Sequence number of last received server message
  uint64 last_svr_seqno = 2;

  repeated DOTSMitigation mitigations = 3;

  // Request active mitigation list from server
  bool active = 4;

  // Ping request (operator initiated)
  bool ping = 5;

  DOTSSessionConfig config = 6;

  repeated google.protobuf.Any extensions = 999;
}
```

Figure 7: DOTS Client Message Schema

7.2. Mitigation Request Schema

```
message DOTSMitigation {
  // Opaque client-generated event identifier
  string eventid = 1;

  // Toggle mitigation for the above scope
  bool requested = 2;

  // Mitigation scope as described in I-D.ietf-dots-requirements
  string scope = 3;

  // Lifetime of the requested mitigation.
  uint32 lifetime = 4;

  // Mitigation efficacy score as a float value between 0 and 1
  float efficacy = 5;

  repeated google.protobuf.Any extensions = 999;
}
```

Figure 8: DOTS Client Mitigation Request Schema

7.3. Session Configuration Schema

```
// Per session configuration sent on signaling session init
message DOTSSessionConfig {
  // Acceptable signal loss
  uint32 loss_limit = 1;

  // Maximum mitigation lifetime in seconds
  uint32 lifetime_max = 2;

  // Heartbeat interval in milliseconds
  uint32 heartbeat_interval = 3;
}
```

Figure 9: DOTS Session Conifiguration Schema

7.4. DOTS Server Message Schema

```
syntax = "proto3";
import "google/protobuf/any.proto";

message DOTSServerMessage {
  // Server generated sequence number
  uint64 seqno = 1;

  // Sequence number of last received Client message
  uint64 last_client_seqno = 2;

  // Request immediate heartbeat response from client.
  bool ping = 3;

  // Server error details, if available
  DOTSServerError error = 4;

  DOTSRedirect redirect = 5;

  // Mitigation data, limited by MTU
  repeated DOTSMitigationStatus mitigations = 6;
}
```

Figure 10: DOTS Server Message Schema

7.5. DOTS Redirect Schema

```
message DOTSRedirect {
  // Redirection target DOTS server address
  string target = 1;

  // Address family of redirection target
  enum RedirectionTargetType {
    DNSNAME = 0;
    IPV4 = 4;
    IPV6 = 6;
```

```
    }
    RedirectionTargetType target_type = 2;

    // Port on which to contact redirection target.
    // XXX Protobufs has no uint16 type, implementations
    // will need to sanity check.
    uint32 port = 3;
  }
```


7.6. DOTS Mitigation Status Schema

```
syntax = "proto3";
import "google/protobuf/any.proto";

message DOTSMitigationStatus {
  // Opaque Client generated event identifier, used by DOTS client
  // to associate a mitigation status with the event triggering the
  // mitigation request.
  string eventid = 1;

  // Mitigation state
  bool enabled = 2;

  // Mitigation time-to-live (lifetime - (now - start))
  uint32 ttl = 3;

  // Dropped byte count
  uint64 bytes_dropped = 4;

  // Dropped bits per second
  uint64 bps_dropped = 5;

  // Dropped packet count
  uint64 pkts_dropped = 6;

  // Dropped packets per second
  uint64 pps_dropped = 7;
```

```
    // Blacklist enabled through data channel
    bool blacklist_enabled = 8;

    // Whitelist enabled through data channel
    bool whitelist_enabled = 9;

    // Filters enabled through data channel
    bool filters_enabled = 10;

    repeated google.protobuf.Any extensions = 999;
  }
```

Figure 11: DOTS Server Mitigation Status Schema

7.7. Server Error Schema


```
syntax = "proto3";
import "google/protobuf/any.proto";

message DOTSServerError {
  enum ErrorCode {
    NOERROR = 0;
    INVALID_VALUE = 1;
    MITIGATION_UNAVAILABLE = 2;
    MITIGATION_CONFLICT = 3;
    MALFORMED_MESSAGE = 4;
  }
  ErrorCode code = 1;

  // Error details, returned as a blob
  google.protobuf.Any details = 2;
}
```

Figure 12: DOTS Server Error Schema

8. References

8.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <http://www.rfc-editor.org/info/rfc768>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

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

[RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <http://www.rfc-editor.org/info/rfc2784>.

[RFC3031] Rosen, E., Viswanathan, A., and R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, DOI 10.17487/RFC3031, January 2001, <http://www.rfc-editor.org/info/rfc3031>.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, DOI 10.17487/RFC5405, November 2008, <http://www.rfc-editor.org/info/rfc5405>.

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

[RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, DOI 10.17487/RFC6555, April 2012, <http://www.rfc-editor.org/info/rfc6555>.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication

of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, DOI 10.17487/RFC6698, August 2012, <http://www.rfc-editor.org/info/rfc6698>.

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

[I-D.ietf-dots-architecture]
Mortensen, A., Andreasen, F., Reddy, T., christopher_gray3@cable.comcast.com, c., Compton, R., and N. Teague, "Distributed-Denial-of-Service Open Threat Signaling (DOTS) Architecture", draft-ietf-dots-architecture-01 (work in progress), October 2016.

[I-D.ietf-dots-requirements]
Mortensen, A., Moskowitz, R., and T. Reddy, "Distributed Denial of Service (DDoS) Open Threat Signaling Requirements", draft-ietf-dots-requirements-03 (work in progress), October 2016.

[PROTOBUF]
Google, Inc., "Protocol Buffers", 2016, <https://developers.google.com/protocol-buffers/>.

8.2. Informative References

[I-D.reddy-dots-data-channel]
Reddy, T., Boucadair, M., Nishizuka, K., Xia, L., and P. Patil, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel", draft-reddy-dots-data-channel-03 (work in progress), December 2016.

[RFC1518] Rekhter, Y. and T. Li, "An Architecture for IP Address Allocation with CIDR", RFC 1518, DOI 10.17487/RFC1518, September 1993, <http://www.rfc-editor.org/info/rfc1518>.

[RFC1519] Fuller, V., Li, T., Yu, J., and K. Varadhan, "Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy", RFC 1519, DOI 10.17487/RFC1519,

September 1993, <http://www.rfc-editor.org/info/rfc1519>.

[RFC5575] Marques, P., Sheth, N., Raszuk, R., Greene, B., Mauch, J., and D. McPherson, "Dissemination of Flow Specification Rules", RFC 5575, DOI 10.17487/RFC5575, August 2009, <http://www.rfc-editor.org/info/rfc5575>.

[CLOUDSIGNAL] Arbor Networks, Inc., "Cloud Signaling: A Faster, Automated Way to Mitigate DDoS Attacks", 2011, <https://www.arbornetworks.com/cloud-signaling-a-faster-automated-way-to-mitigate-ddos-attacks>.

[COMMUNITYFS] Team Cymru, Inc., "Community FlowSpec", 2011, <https://www.cymru.com/jtk/misc/community-fs.html>.

[OPENHYBRID] Verisign, Inc., "Verisign OpenHybrid", 2016, <http://www.verisign.com/en_US/security-services/ddos-protection/open-api/index.xhtml>.

[WISR] Arbor Networks, Inc., "Worldwide Infrastructure Security Report", 2016, <https://www.arbornetworks.com/images/documents/WISR2016_EN_Web.pdf>.

Authors' Addresses

Nik Teague
Verisign, Inc.
12061 Bluemont Way
Reston, VA 20190
United States

Email: nteague@verisign.com

Andrew Mortensen
Arbor Networks, Inc.
2727 S. State St
Ann Arbor, MI 48104
United States

Email: amortensen@arbor.net