DOTS ~~T. Reddy,~~ **M. Boucadair,** Ed.
Internet-Draft ~~McAfee~~ **Orange**
Intended status: Standards Track ~~M. Boucadair,~~ **T. Reddy,** Ed.
Expires: January ~~18,~~ **28,** 2019 ~~Orange~~ **McAfee**
K. Nishizuka
NTT Communications
L. Xia
Huawei
P. Patil
Cisco
A. Mortensen
Arbor Networks, Inc.
N. Teague
Verisign, Inc.
July ~~17,~~ **27,** 2018

Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel Specification

~~draft-ietf-dots-data-channel-17~~
draft-ietf-dots-data-channel-18

Abstract

The document specifies a Distributed Denial-of-Service Open Threat Signaling (DOTS) data channel used for bulk exchange of data that cannot easily or appropriately communicated through the DOTS signal channel under attack conditions.

This is a companion document to the DOTS signal channel specification.

Editorial Note (To be removed by RFC Editor)

Please update these statements **within the document** with the RFC number to be assigned to this document:

- o "This version of this YANG module is part of RFC XXXX;"
- o "RFC XXXX: Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel Specification";
- o reference: RFC XXXX

Please update these statements with the RFC number to be assigned to the following documents:

- o "RFC YYYY: Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification" (used to be [I-D.ietf-dots-signal-channel])
- o "RFC ZZZZ: Network Access Control List (ACL) YANG Data Model" (used to be [I-D.ietf-netmod-acl-model])

Please update the "revision" date of the YANG module.

Table of Contents

1. Introduction

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

As discussed in [I-D.ietf-dots-requirements], the lack of a common method to coordinate a real-time response among involved actors and network domains inhibits the speed and effectiveness of DDoS attack mitigation. From that standpoint, DDoS Open Threat Signaling (DOTS) defines an architecture that allows a DOTS client to send requests to a DOTS server for DDoS attack mitigation [I-D.ietf-dots-architecture]. The DOTS approach is thus meant to minimize the impact of DDoS attacks, thereby contributing to the enforcement of more efficient defensive if not proactive security strategies. To that aim, DOTS defines two channels: the signal and the data channels (Figure 1).

```
+---------------+                                 +---------------+
|               | <------- Signal Channel ------> |               |
|  DOTS Client  |                                 |  DOTS Server  |
|               | <=======  Data Channel  ======> |               |
+---------------+                                 +---------------+
```

Figure 1: DOTS Channels

The DOTS signal channel is used to carry information about a device or a network (or a part thereof) that is under a DDoS attack. Such information is sent by a DOTS client to an upstream DOTS server so that appropriate mitigation actions are undertaken on traffic deemed suspicious. The DOTS signal channel is further elaborated in [I-D.ietf-dots-signal-channel].

As for the DOTS data channel, it is used for infrequent bulk data exchange between DOTS agents to significantly improve the coordination of all the parties involved in the response to the attack. Section 2 of [I-D.ietf-dots-architecture] mentions that the DOTS data channel is used to perform the following tasks:

o Creating aliases for resources for which mitigation may be requested.

  A DOTS client may submit to its DOTS server a collection of prefixes which it would like to refer to by an alias when requesting mitigation. The DOTS server can respond to this request with either a success or failure response (see Section 2 in [I-D.ietf-dots-architecture]).

  Refer to Section 6 for more details.

o Filter management, which enables a DOTS client to request the installation or withdrawal of traffic filters, dropping or rate-limiting unwanted traffic, and permitting white-listed traffic. A DOTS client is entitled to instruct filtering rules only on IP resources that belong to its domain.

  Sample use cases for populating black- or white-list filtering rules are detailed hereafter:

  * If a network resource (DOTS client) detects a potential DDoS attack from a set of IP addresses, the DOTS client informs its servicing DOTS gateway of all suspect IP addresses that need to be blocked or black-listed for further investigation. The DOTS client could also specify a list of protocols and port numbers in the black-list rule.

    The DOTS gateway then propagates the black-listed IP addresses to a DOTS server which will undertake appropriate actions so that traffic originated by these IP addresses to the target network (specified by the DOTS client) is blocked.

* A network, that has partner sites from which only legitimate traffic arrives, may want to ensure that the traffic from these sites is not subjected to DDoS attack mitigation. The DOTS client uses the DOTS data channel to convey the white-listed IP prefixes of the partner sites to its DOTS server.

  The DOTS server uses this information to white-list flows originated by such IP prefixes and which reach the network.

Refer to Section 7 for more details.

2. Terminology

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

The reader should be familiar with the terms defined in [I-D.ietf-dots-requirements].

The terminology for describing YANG data modules is defined in [RFC7950]. The meaning of the symbols in tree diagrams is defined in [RFC8340].

This document generalizes the notion of Access Control List (ACL) so that it is not device-specific [I-D.ietf-netmod-acl-model]. As such, this document defines an ACL as an ordered set of rules that is used to filter traffic. Each rule is represented by an Access Control Entry (ACE). ACLs communicated via the DOTS data channel are not bound to a device interface.

For the sake of simplicity, all of the examples in this document use "/restconf" as the discovered RESTCONF API root path. Many protocol header lines and message-body text within examples throughout the document are split into multiple lines for display purposes only. When a line ends with backslash ('\') as the last character, the line is wrapped for display purposes. It is to be considered to be joined to the next line by deleting the backslash, the following line break, and the leading whitespace of the next line.

3. DOTS Data Channel

3.1. Design Overview

Unlike the DOTS signal channel, which must remain operational even when confronted with signal degradation due to packets loss, the DOTS data channel is not expected to be fully operational at all times, especially when a DDoS attack is underway. The requirements for a DOTS data channel protocol are documented in [I-D.ietf-dots-requirements].

This specification does not require an order of DOTS signal and data channel creations nor mandates a time interval between them. These considerations are implementation- and deployment-specific.

As the primary function of the data channel is data exchange, a reliable transport mode is required in order for DOTS agents to detect data delivery success or failure. This document uses RESTCONF [RFC8040] over TLS [RFC5246] over TCP as the DOTS data channel protocol. The abstract layering of DOTS data channel is shown in Figure 2.

```
+-------------------+
| DOTS Data Channel |
+-------------------+
|      RESTCONF     |
+-------------------+
|        TLS        |
+-------------------+
|        TCP        |
+-------------------+
|        IP         |
+-------------------+
```

Figure 2: Abstract Layering of DOTS Data Channel

The HTTP POST, PUT, PATCH, and DELETE methods are used to edit data resources represented by DOTS data channel YANG data modules. These basic edit operations allow the DOTS data channel running configuration to be altered by a DOTS client.

DOTS data channel configuration information as well as state information can be retrieved with the GET method. An HTTP status-line header field is returned for each request to report success or failure for RESTCONF operations (Section 5.4 of [RFC8040]). The "error-tag" provides more information about encountered errors (Section 7 of [RFC8040]).

DOTS clients perform the root resource discovery procedure discussed in Section 3.1 of [RFC8040] to determine the root of the RESTCONF API. After discovering the RESTCONF API root, a DOTS client uses this value as the initial part of the path in the request URI, in any subsequent request to the DOTS server. The DOTS server may support the retrieval of the YANG modules it supports (Section 3.7 in

[RFC8040]). For example, a DOTS client may use RESTCONF to retrieve the vendor-specific YANG modules supported by its DOTS server.

JavaScript Object Notation (JSON) [RFC8259] payload is used to propagate the DOTS data channel specific payload messages that carry request parameters and response information, such as errors. This specification uses the encoding rules defined in [RFC7951] for representing DOTS data channel configuration data using YANG (Section 4) as JSON text.

A DOTS client registers itself to its DOTS server(s) in order to set up DOTS data channel-related configuration data and receive state data (i.e., non-configuration data) from the DOTS server(s) (Section 5). Mutual authentication and coupling of signal and data channels are specified in [I-D.ietf-dots-signal-channel].

A single DOTS data channel between DOTS agents can be used to exchange multiple requests and multiple responses. To reduce DOTS client and DOTS server workload, DOTS clients SHOULD re-use the same TLS session. While the communication to the DOTS server is quiescent, the DOTS client MAY probe the server to ensure it has maintained cryptographic state. Such probes can also keep alive firewall and/or NAT bindings. A TLS heartbeat [RFC6520] verifies that the DOTS server still has TLS state by returning a TLS message.

A DOTS server may detect conflicting filtering requests from distinct DOTS clients which belong to the same domain. For example, a DOTS client could request to blacklist a prefix by specifying the source prefix, while another DOTS client could request to whitelist that same source prefix, but both having the same destination prefix. It is out of scope of this specification to recommend the behavior to follow for handling conflicting requests (e.g., reject all, reject the new request, notify an administrator for validation). DOTS servers SHOULD support a configuration parameter to indicate the behavior to follow when a conflict is detected. Section 7.2 specifies the behavior when no instruction is supplied to a DOTS server.

How filtering rules instantiated on a DOTS server are translated into network configurations actions is out of scope.

3.2. DOTS Server(s) Discovery

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

Likewise, it is out of scope of this document to specify the behavior to ~~follow~~ **be followed** by a DOTS client to ~~place its~~ **send DOTS** requests **when multiple DOTS servers are provisioned** (e.g., contact all **DOTS** servers, select one **DOTS** server among the ~~list) when multiple DOTS servers are provisioned.~~ **list).**

3.3. NAT Considerations

In deployments where one or more translators (e.g., NAT44, NAT64, NPTv6) are enabled between the client's network and the DOTS server, DOTS data channel messages forwarded to a DOTS server ~~must not~~ **MUST NOT** include internal IP addresses/prefixes and/or port numbers; external addresses/prefixes and/or port numbers as assigned by the translator MUST be used instead. This document does not make any recommendation 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) [RFC5389] may be used 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 data 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.

3.4. DOTS Gateways

When a server-domain DOTS gateway is involved in DOTS data channel exchanges, the same considerations for manipulating the 'cdid' (client domain identifier) parameter specified in [I-D.ietf-dots-signal-channel] MUST be followed by DOTS agents. As a reminder, 'cdid' is meant to assist the DOTS server to enforce some policies (e.g., limit the number of filtering rules per DOTS client or per DOTS client domain). A loop detect mechanism for DOTS gateways is specified in Section 3.5.

If a DOTS gateway is involved, the DOTS gateway verifies that the DOTS client is authorized to undertake a data channel action (e.g., instantiate filtering rules). If the DOTS client is authorized, it propagates the rules to the upstream DOTS server. Likewise, the DOTS server verifies that the DOTS gateway is authorized to relay data channel actions. For example, to create or purge filters, a DOTS client sends its request to its DOTS gateway. The DOTS gateway validates the rules in the request and proxies the requests containing the filtering rules to its DOTS server. When the DOTS gateway receives the associated response from the DOTS server, it propagates the response back to the DOTS client.

3.5. Detect and Prevent Infinite Loops

In order to detect and prevent infinite loops, DOTS gateways MUST support the procedure defined in Section 5.7.1 of [RFC7230]. In particular, each intermediate DOTS gateway MUST check that none of its own information (e.g., server names, literal IP addresses) is present in the "Via" header of a DOTS message it receives:

- o If it detects that its own information is present in the "Via" header, the DOTS gateway MUST NOT forward the DOTS message. Messages that cannot be forwarded because of a loop SHOULD be logged with a "508 Loop Detected" status-line returned sent back to the DOTS peer. The structure of the reported error is depicted in Figure 3.

```
error-tag:      loop-detected
error-type:     transport, application
error-severity: error
error-info:     <via-header> : A copy of the Via header when
                the loop was detected.
Description:    An infinite loop has been detected when forwarding
                a requests via a proxy.
```

Figure 3: Loop Detected Error

  It is RECOMMENDED that DOTS clients and gateways support means to alert administrators about loop errors so that appropriate actions are undertaken.

- o Otherwise, the DOTS agent MUST update or insert the "Via" header by appending its own information.

Unless configured otherwise, DOTS gateways at the boundaries of a DOTS client domain SHOULD remove the previous "Via" header information after checking for a loop before forwarding. This behavior is required for topology hiding purposes but also to minimizing potential conflicts that may arise if overlapping information is used in distinct DOTS domains (e.g., private IPv4 addresses, non globally unique aliases).

3.6. Stale Entries

In order to avoid stale entries, a lifetime is associated with alias and filtering entries created by DOTS clients. Also, DOTS servers may track the inactivity timeout of DOTS clients to detect stale entries.

4. DOTS Data Channel YANG Module

4.1. Tree Structure

The DOTS data channel YANG module (ietf-dots-data-channel) allows a DOTS client to manage aliases for resources for which mitigation may be requested. Such aliases may be used in subsequent DOTS signal channel exchanges to refer more efficiently to the resources under attack.

The tree structure for the DOTS alias is depicted in Figure 4.

```
module: ietf-dots-data-channel
    +--rw dots-data
       +--rw dots-client* [cuid]
       |  +--rw cuid            string
       |  +--rw cdid?           string
       |  +--rw aliases
       |  |  +--rw alias* [name]
       |  |     +--rw name                 string
       |  |     +--rw target-prefix*       inet:ip-prefix
       |  |     +--rw target-port-range* [lower-port upper-port]
       |  |     |  +--rw lower-port    inet:port-number
       |  |     |  +--rw upper-port    inet:port-number
       |  |     +--rw target-protocol*     uint8
       |  |     +--rw target-fqdn*         inet:domain-name
       |  |     +--rw target-uri*          inet:uri
       |  |     +--ro pending-lifetime?    int32
       |  +--rw acls
       |     ...
       +--ro capabilities
          ...
```

Figure 4: DOTS Alias Subtree

Also, the 'ietf-dots-data-channel' module allows DOTS clients to manage filtering rules. Examples of filtering management in a DOTS context include, but not limited to:

- o Black-list management, which enables a DOTS client to inform a DOTS server about sources from which traffic should be discarded.
- o White-list management, which enables a DOTS client to inform a DOTS server about sources from which traffic should always be accepted.
- o Filter management, which enables a DOTS client to request the installation or withdrawal of traffic filters, dropping or rate-limiting unwanted traffic and permitting white-listed traffic.

The tree structure for the DOTS filtering entries is depicted in

Figure 5.

Early versions of this document investigated to what extent augmenting 'ietf-access-control-list' meet DOTS requirements, but that design approach was abandoned because it does not support meeting many of DOTS requirements, e.g.,

- o Retrieve a filtering entry (or all entries) created by a DOTS client.
- o Delete a filtering entry that was instantiated by a DOTS client.

DOTS filtering entries (i.e., Access Control List (ACL)) mimic the structure specified in [I-D.ietf-netmod-acl-model]. Concretely, DOTS agents are assumed to manipulate an ordered list of ACLs; each ACL contains a separately ordered list of Access Control Entries (ACEs). Each ACE has a group of match and a group of action criteria.

Once all the ACE entries have been iterated though with no match, then all the following ACL's ACE entries are iterated through until the first match at which point the specified action is applied. If there is no match, then there is no action to be taken against the packet.

```
module: ietf-dots-data-channel
    +--rw dots-data
       +--rw dots-client* [cuid]
       |  +--rw cuid            string
       |  +--rw cdid?           string
       |  +--rw aliases
       |  |  ...
       |  +--rw acls
       |     +--rw acl* [name]
       |        +--rw name                string
       |        +--rw type?               ietf-acl:acl-type
       |        +--rw activation-type?    enumeration
       |        +--ro pending-lifetime?   int32
       |        +--rw aces
       |           +--rw ace* [name]
       |              +--rw name          string
       |              +--rw matches
       |              |  +--rw (l3)?
       |              |  |  +--:(ipv4)
       |              |  |  |  ...
       |              |  |  +--:(ipv6)
       |              |  |     ...
       |              |  +--rw (l4)?
       |              |     +--:(tcp)
       |              |     |  ...
       |              |     +--:(udp)
       |              |     |  ...
       |              |     +--:(icmp)
       |              |        ...
       |              +--rw actions
       |              |  +--rw forwarding    identityref
       |              |  +--rw rate-limit?   decimal64
       |              +--ro statistics
       |                 +--ro matched-packets?   yang:counter64
       |                 +--ro matched-octets?    yang:counter64
       +--ro capabilities
          ...
```

Figure 5: DOTS ACLs Subtree

Filtering rules instructed by a DOTS client assumes a default direction: the destination is the DOTS client domain.

DOTS forwarding actions can be 'accept' (i.e., accept matching traffic) or 'drop' (i.e., drop matching traffic without sending any ICMP error message). Accepted traffic can be subject to rate limiting 'rate-limit'. Note that 'reject' action (i.e., drop matching traffic and send an ICMP error message to the source) is not supported in 'ietf-dots-data-channel' because it is not appropriate in the context of DDoS mitigation. Generating ICMP messages to notify drops when mitigating a DDoS attack will exacerbate the DDoS attack. Furthermore, these ICMP messages will be used by an attacker as an explicit signal that the traffic is being blocked.

4.2. Filtering Fields

The 'ietf-dots-data-channel' module reuses the packet fields module 'ietf-packet-fields' [I-D.ietf-netmod-acl-model] which defines matching on fields in the packet including IPv4, IPv6, and transport layer fields.

This specification defines a new IPv4/IPv6 matching field called 'fragment' to efficiently handle fragment-related filtering rules. Indeed, [I-D.ietf-netmod-acl-model] does not support such capability for IPv6 but offers a partial support for IPv4 by means of 'flags'. Nevertheless, the use of 'flags' is problematic since it does not allow to define a bitmask. For example, setting other bits not covered by the 'flags' filtering clause in a packet will allow that packet to get through (because it won't match the ACE). Sample examples to illustrate how 'fragment' can be used are provided in Appendix A.

Figure 6 shows the IPv4 match subtree.

```
module: ietf-dots-data-channel
    +--rw dots-data
       +--rw dots-client* [cuid]
       |  ...
       |  +--rw acls
       |     +--rw acl* [name]
       |        ...
       |        +--rw aces
       |           +--rw ace* [name]
       |              +--rw name          string
       |              +--rw matches
       |              |  +--rw (l3)?
       |              |  |  +--:(ipv4)
       |              |  |  |  +--rw ipv4
       |              |  |  |     +--rw dscp?                 inet:dscp
       |              |  |  |     +--rw ecn?                  uint8
       |              |  |  |     +--rw length?               uint16
       |              |  |  |     +--rw ttl?                  uint8
       |              |  |  |     +--rw protocol?             uint8
       |              |  |  |     +--rw ihl?                  uint8
       |              |  |  |     +--rw flags?                bits
       |              |  |  |     +--rw offset?               uint16
       |              |  |  |     +--rw identification?       uint16
       |              |  |  |     +--rw (destination-network)?
       |              |  |  |     |  +--:(destination-ipv4-network)
       |              |  |  |     |     +--rw destination-ipv4-network?
       |              |  |  |     |             inet:ipv4-prefix
       |              |  |  |     +--rw (source-network)?
       |              |  |  |     |  +--:(source-ipv4-network)
       |              |  |  |     |     +--rw source-ipv4-network?
       |              |  |  |     |             inet:ipv4-prefix
       |              |  |  |     +--rw fragment
       |              |  |  |        +--rw operator?        operator
       |              |  |  |        +--rw type        fragment-type
       |              |  |  +--:(ipv6)
       |              |  |     ...
       |              |  +--rw (l4)?
       |              |     ...
       |              +--rw actions
       |              |  ...
       |              +--ro statistics
       |                 ...
       +--ro capabilities
          ...
```

Figure 6: DOTS ACLs Subtree (IPv4 Match)

Figure 7 shows the IPv6 match subtree.

```
module: ietf-dots-data-channel
    +--rw dots-data
       +--rw dots-client* [cuid]
       |  ...
       |  +--rw acls
       |     +--rw acl* [name]
       |        ...
       |        +--rw aces
       |           +--rw ace* [name]
       |              +--rw name          string
       |              +--rw matches
       |              |  +--rw (l3)?
       |              |  |  +--:(ipv4)
       |              |  |  |  ...
       |              |  |  +--:(ipv6)
       |              |  |     +--rw ipv6
       |              |  |        +--rw dscp?                 inet:dscp
       |              |  |        +--rw ecn?                  uint8
       |              |  |        +--rw length?               uint16
       |              |  |        +--rw ttl?                  uint8
       |              |  |        +--rw protocol?             uint8
       |              |  |        +--rw (destination-network)?
       |              |  |        |  +--:(destination-ipv6-network)
       |              |  |        |     +--rw destination-ipv6-network?
       |              |  |        |             inet:ipv6-prefix
       |              |  |        +--rw (source-network)?
       |              |  |        |  +--:(source-ipv6-network)
       |              |  |        |     +--rw source-ipv6-network?
       |              |  |        |             inet:ipv6-prefix
       |              |  |        +--rw flow-label?
       |              |  |        |       inet:ipv6-flow-label
       |              |  |        +--rw ~~fragment?             empty~~ fragment
       |              |  |           +--rw operator?       operator
       |              |  |           +--rw type       fragment-type
       |              |  +--rw (l4)?
       |              |     ...
       |              +--rw actions
       |              |  ...
       |              +--ro statistics
       |                 ...
       +--ro capabilities
          ...
```

Figure 7: DOTS ACLs Subtree (IPv6 Match)

Figure 8 shows the TCP match subtree. **In addition to the fields defined in [I-D.ietf-netmod-acl-model], this specification defines a new TCP matching field, called 'flags-bitmask', to efficiently handle TCP flags filtering rules.**

```
module: ietf-dots-data-channel
    +--rw dots-data
       +--rw dots-client* [cuid]
       |  ...
       |  +--rw acls
       |     +--rw acl* [name]
       |        ...
       |        +--rw aces
       |           +--rw ace* [name]
       |              +--rw name          string
       |              +--rw matches
       |              |  +--rw (l3)?
       |              |  |  ...
       |              |  +--rw (l4)?
       |              |     +--:(tcp)
       |              |     |  +--rw tcp
       |              |     |     +--rw sequence-number?          uint32
       |              |     |     +--rw acknowledgement-number?   uint32
       |              |     |     +--rw data-offset?              uint8
       |              |     |     +--rw reserved?                 uint8
       |              |     |     +--rw flags?                    bits
       |              |     |     +--rw window-size?              uint16
       |              |     |     +--rw urgent-pointer?           uint16
       |              |     |     +--rw options?                  uint32
       |              |     |     +--rw flags-bitmask
       |              |     |     |  +--rw operator?            operator
       |              |     |     |  +--rw bitmask                uint16
       |              |     |     +--rw (source-port)?
       |              |     |     |  +--:(source-port-range-or-operator)
       |              |     |     |     +--rw source-port-range-or-operator
       |              |     |     |        +--rw (port-range-or-operator)?
       |              |     |     |           +--:(range)
       |              |     |     |           |  +--rw lower-port
       |              |     |     |           |  |       inet:port-number
       |              |     |     |           |  +--rw upper-port
       |              |     |     |           |          inet:port-number
       |              |     |     |           +--:(operator)
       |              |     |     |              +--rw operator?
       |              |     |     |              |       operator
       |              |     |     |              +--rw port
       |              |     |     |                      inet:port-number
       |              |     |     +--rw (destination-port)?
       |              |     |        +--:(destination-port-range-or-operator)
       |              |     |           +--rw destination-port-range-or-operator
       |              |     |              +--rw (port-range-or-operator)?
       |              |     |                 +--:(range)
       |              |     |                 |  +--rw lower-port
       |              |     |                 |  |       inet:port-number
       |              |     |                 |  +--rw upper-port
       |              |     |                 |          inet:port-number
       |              |     |                 +--:(operator)
       |              |     |                    +--rw operator?
       |              |     |                    |       operator
       |              |     |                    +--rw port
       |              |     |                            inet:port-number
       |              |     +--:(udp)
       |              |     |  ...
       |              |     +--:(icmp)
       |              |        ...
       |              +--rw actions
       |              |  ...
       |              +--ro statistics
       |                 ...
       +--ro capabilities
          ...
```

Figure 8: DOTS ACLs Subtree (TCP Match)

Figure 9 shows the UDP and ICMP match subtree.

```
module: ietf-dots-data-channel
    +--rw dots-data
       +--rw dots-client* [cuid]
       |  ...
       |  +--rw acls
       |     +--rw acl* [name]
       |        ...
       |        +--rw aces
       |           +--rw ace* [name]
       |              +--rw name          string
       |              +--rw matches
       |              |  +--rw (l3)?
       |              |  |  ...
       |              |  +--rw (l4)?
       |              |     +--:(tcp)
       |              |     |  ...
       |              |     +--:(udp)
       |              |     |  +--rw udp
       |              |     |     +--rw length?          uint16
       |              |     |     +--rw (source-port)?
       |              |     |     |  +--:(source-port-range-or-operator)
       |              |     |     |     +--rw source-port-range-or-operator
       |              |     |     |        +--rw (port-range-or-operator)?
       |              |     |     |           +--:(range)
       |              |     |     |           |  +--rw lower-port
       |              |     |     |           |  |       inet:port-number
       |              |     |     |           |  +--rw upper-port
```

```
     |              |     |     |           |          inet:port-number
     |              |     |     |           +--:(operator)
     |              |     |     |              +--rw operator?
     |              |     |     |              |       operator
     |              |     |     |              +--rw port
     |              |     |     |                      inet:port-number
     |              |     |     +--rw (destination-port)?
     |              |     |        +--:(destination-port-range-or-operator)
     |              |     |           +--rw destination-port-range-or-operator
     |              |     |              +--rw (port-range-or-operator)?
     |              |     |                 +--:(range)
     |              |     |                 |  +--rw lower-port
     |              |     |                 |  |       inet:port-number
     |              |     |                 |  +--rw upper-port
     |              |     |                 |          inet:port-number
     |              |     |                 +--:(operator)
     |              |     |                    +--rw operator?
     |              |     |                    |       operator
     |              |     |                    +--rw port
     |              |     |                            inet:port-number
     |              |     +--:(icmp)
     |              |        +--rw icmp
     |              |           +--rw type?             uint8
     |              |           +--rw code?             uint8
     |              |           +--rw rest-of-header?   uint32
     |              +--rw actions
     |              |  ...
     |              +--ro statistics
     |                 ...
     +--ro capabilities
        ...
```

Figure 9: DOTS ACLs Subtree (UDP and ICMP Match)

DOTS implementations MUST support the following matching criteria:

match based on the IP header (IPv4 and IPv6), match based on the transport header (TCP, UDP, and ICMP), and any combination thereof. The same matching fields are used for both ICMP and ICMPv6.

The following match fields MUST be supported by DOTS implementations (Table 1):

ACL Match	Mandatory Fields
ipv4	length, protocol, ~~flags,~~ destination-ipv4-network, **source-ipv4-network,** and ~~source-ipv4-network~~ **fragment**
ipv6	length, protocol, destination-ipv6-network, source-ipv6-network, and fragment
tcp	~~flags,~~ **flags-bitmask,** source-port-range-or-operator, and ~~destination-port-range-or-operator~~ **destination-port-range-or-operator**
udp	length, source-port-range-or-operator, and destination-port-range-or-operator
icmp	type and code

Table 1: Mandatory DOTS Channel Match Fields

Implementations MAY support other filtering match fields and actions. The 'ietf-dots-data-channel' allows an implementation to expose its filtering capabilities. The tree structure of the 'capabilities' is shown in Figure 10.

```
module: ietf-dots-data-channel
    +--rw dots-data
       ...
       +--ro capabilities
          +--ro address-family*        enumeration
          +--ro forwarding-actions*    identityref
          +--ro rate-limit?            boolean
          +--ro transport-protocols*   uint8
          +--ro ipv4
          |  +--ro dscp?                 boolean
          |  +--ro ecn?                  boolean
          |  +--ro length?               boolean
          |  +--ro ttl?                  boolean
          |  +--ro protocol?             boolean
          |  +--ro ihl?                  boolean
          |  +--ro flags?                boolean
          |  +--ro offset?               boolean
          |  +--ro identification?       boolean
          |  +--ro source-prefix?        boolean
          |  +--ro destination-prefix?   boolean
          |  +--ro ~~fragment~~ fragment?             boolean
          +--ro ipv6
          |  +--ro dscp?                 boolean
          |  +--ro ecn?                  boolean
          |  +--ro flow-label?           boolean
          |  +--ro length?               boolean
          |  +--ro protocol?             boolean
          |  +--ro hoplimit?             boolean
          |  +--ro source-prefix?        boolean
          |  +--ro destination-prefix?   boolean
          |  +--ro ~~fragment~~ fragment?             boolean
```

```
           +--ro tcp
           |  +--ro sequence-number?          boolean
           |  +--ro acknowledgement-number?   boolean
           |  +--ro data-offset?              boolean
           |  +--ro reserved?                 boolean
           |  +--ro flags?                    boolean
           |  +--ro flags-bitmask?            boolean
           |  +--ro window-size?              boolean
           |  +--ro urgent-pointer?           boolean
           |  +--ro options?                  boolean
           |  +--ro source-port?              boolean
           |  +--ro destination-port?         boolean
           |  +--ro port-range?               boolean
           +--ro udp
           |  +--ro length?             boolean
           |  +--ro source-port?        boolean
           |  +--ro destination-port?   boolean
           |  +--ro port-range?         boolean
           +--ro icmp
              +--ro type?             boolean
              +--ro code?             boolean
              +--ro rest-of-header?   boolean

              Figure 10: Filtering Capabilities Sub-Tree
```

4.3. YANG Module

```
  <CODE BEGINS> file ~~"ietf-dots-data-channel@2018-05-15.yang"~~ "ietf-dots-data-channel@2018-07-25.yang"

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

    import ietf-access-control-list {
      prefix ietf-acl;
    }
    import ietf-packet-fields {
      prefix packet-fields;
    }
    import ietf-dots-signal-channel {
      prefix dots-signal;
    }

    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:  Konda, Tirumaleswar Reddy
                <mailto:TirumaleswarReddy_Konda@McAfee.com>

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

       Author:  Jon Shallow
                <mailto:jon.shallow@nccgroup.trust>

       Author:  Kaname Nishizuka
                <mailto:kaname@nttv6.jp>

       Author:  Liang Xia
                <mailto:frank.xialiang@huawei.com>

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

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

       Author:  Nik Teague
                ~~<mailto:nteague@verisign.com>~~

       ~~Author:  Jon Shallow~~
                ~~<mailto:jon.shallow@nccgroup.trust>";~~
                <mailto:nteague@verisign.com>";
    description
      "This module contains YANG definition for configuring
       aliases for resources and filtering rules using DOTS
       data channel.

       Copyright (c) 2018 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 XXXX; see
       the RFC itself for full legal notices.";
```

```
     revision 2018-05-15 2018-07-25 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: Distributed Denial-of-Service Open Threat
                    Signaling (DOTS) Data Channel Specification";
     }
     grouping aliases

     typedef operator {
       description
         "Top level
       type bits {
         bit not {
           position 0;
           description
             "If set, logical negation of operation.";
         }
         bit match {
           position 1;
           description
             "Match bit.  If set, this is a bitwise match operation
              defined as '(data & value) == value'; if unset, (data &
              value) evaluates to TRUE if any of the bits in the value
              mask are set in the data.";
         }
       }
       description
         "How to apply the defined bitmask.";
     }

     grouping tcp-flags {
       leaf operator {
         type operator;
         default match;
         description
           "How to interpret the TCP flags.";
       }
       leaf bitmask {
         type uint16;
         mandatory true;
         description
           "Bitmask values can be encoded as a 1- or 2-byte bitmask.
            When a single byte is specified, it matches byte 13
            of the TCP header, which contains bits 8 though 15
            of the 4th 32-bit word.  When a 2-byte encoding is used,
            it matches bytes 12 and 13 of the TCP header with
            the data offset field having a 'don't care' value.";
       }
       description
         "Operations on TCP flags.";
     }
     typedef fragment-type {
       type bits {
         bit df {
           position 0;
           description
             "Don't fragment bit for IPv4.
              This bit must be set to 0 for IPv6.";
         }
         bit isf {
           position 1;
           description
             "Is a fragment.";
         }
         bit ff {
           position 2;
           description
             "First fragment.";
         }
         bit lf {
           position 3;
           description
             "Last fragment.";
         }
       }
       description
         "Different fragment types to match against.";
     }

     grouping fragment-fields {
       leaf operator {
         type operator;
         default match;
         description
           "How to interpret the fragment type.";
       }
       leaf type {
         type fragment-type;
         mandatory true;
         description
           "What fragment type to look for.";
       }
       description
         "Operations on fragment types.";
     }

     grouping aliases {
```

```
    description
      "Top level container for aliases";

    list alias {
      key "name";
      description
        "List of aliases";
      leaf name {
        type string;
        description
          "The name of the alias";
      }
      uses dots-signal:target;
      leaf pending-lifetime {
        type int32;
        units "minutes";
        config false;
        description
          "Indicates the pending validity lifetime of the alias
           entry.";
      }
    }
  }

  grouping ports {
    choice source-port {
      container source-port-range-or-operator {
        uses packet-fields:port-range-or-operator;
        description
          "Source port definition.";
      }
      description
        "Choice of specifying the source port or referring to
         a group of source ports.";
    }
    choice destination-port {
      container destination-port-range-or-operator {
        uses packet-fields:port-range-or-operator;
        description
          "Destination port definition.";
      }
      description
        "Choice of specifying a destination port or referring
         to a group of destination ports.";
    }
    description
      "Choice of specifying a source or destination ports.";
  }

  grouping access-lists {
    description
      "Specifies the ordered set of Access Control Lists.";
    list acl {
      key "name";
      ordered-by user;
      description
        "An Access Control List (ACL) is an ordered list of
         Access Control Entries (ACE). Each Access Control Entry
         has a list of match criteria and a list of actions.";
      leaf name {
        type string {
          length "1..64";
        }
        description
          "The name of the access list.";
        reference
           "RFC ZZZZ: Network Access Control List (ACL)
                      YANG Data Model";
      }
      leaf type {
        type ietf-acl:acl-type;
        description
          "Type of access control list. Indicates the primary intended
           type of match criteria (e.g., IPv4, IPv6) used in the list
           instance.";
        reference
           "RFC ZZZZ: Network Access Control List (ACL)
                      YANG Data Model";
      }
      leaf activation-type {
        type enumeration {
          enum "activate-when-mitigating" {
            value 1;
            description
              "The ACL is installed only when a mitigation is active.
               The ACL is specific to this DOTS client.";
          }
          enum "immediate" {
            value 2;
            description
              "The ACL is immediately activated.";
          }
        }
        description
          "Indicates whether an ACL is to be installed immediately
           or when a mitigation is active.";
      }
      leaf pending-lifetime {
```

```
          type int32;
          units "minutes";
          config false;
          description
            "Indicates the pending validity lifetime of the alias
             entry.";
        }
        container aces {
          description
            "The Access Control Entries container contains
             a list of ACEs.";
          list ace {
            key "name";
            ordered-by user;
            description
              "List of access list entries.";
            leaf name {
              type string {
                length "1..64";
              }
              description
                "A unique name identifying this Access List
                 Entry (ACE).";
              reference
                "RFC ZZZZ: Network Access Control List (ACL)
                           YANG Data Model";
            }
            container matches {
              description
                "The rules in this set determine what fields will be
                 matched upon before any action is taken on them.

                 If no matches are defined in a particular container,
                 then any packet will match that container.

                 If no matches are specified at all in an ACE, then any
                 packet will match the ACE.";
              reference
                 "RFC ZZZZ: Network Access Control List (ACL)
                            YANG Data Model";

              choice l3 {
                container ipv4 {
                  when "derived-from(../../../../type," +
                       "'ietf-acl:ipv4-acl-type')";
                  uses packet-fields:acl-ip-header-fields;
                  uses packet-fields:acl-ipv4-header-fields;
                  container fragment {
                    description
                      "Indicates how to handle IPv4 fragments.";
                    uses fragment-fields;
                  }
                  description
                    "Rule set that matches IPv4 header.";
                }
                container ipv6 {
                  when "derived-from(../../../../type," +
                       "'ietf-acl:ipv6-acl-type')";
                  uses packet-fields:acl-ip-header-fields;
                  uses packet-fields:acl-ipv6-header-fields;
                  ~~leaf~~
                  container fragment {
                    ~~type empty;~~
                    description
                      ~~"Handle~~
                      "Indicates how to handle IPv6 ~~fragments. When this keyword~~
                       ~~is present, the match is about assessing~~
                       ~~whether a packet is a fragment (that is,~~
                       ~~a Fragment header is present).";~~ fragments.";
                    uses fragment-fields;
                  }
                  description
                    "Rule set that matches IPv6 header.";
                }
                description
                  "Either IPv4 or IPv6.";
              }
              choice l4 {
                container tcp {
                  uses packet-fields:acl-tcp-header-fields;
                  container flags-bitmask {
                    description
                      "Indicates how to handle TCP flags.";
                    uses tcp-flags;
                  }
                  uses ports;
                  description
                    "Rule set that matches TCP header.";
                }
                container udp {
                  uses packet-fields:acl-udp-header-fields;
                  uses ports;
                  description
                    "Rule set that matches UDP header.";
                }
                container icmp {
                  uses packet-fields:acl-icmp-header-fields;
                  description
```

```
                  "Rule set that matches ICMP/ICMPv6 header.";
              }
              description
                "Can be TCP, UDP, or ICMP/ICMPv6";
            }

          }
          container actions {
            description
              "Definitions of action for this ACE.";
            leaf forwarding {
              type identityref {
                base ietf-acl:forwarding-action;
              }
              mandatory true;
              description
                "Specifies the forwarding action per ACE.";
              reference
                 "RFC ZZZZ: Network Access Control List (ACL)
                            YANG Data Model";
            }
            leaf rate-limit {
              when "../forwarding = 'ietf-acl:accept'" {
                description
                  "rate-limit valid only when accept action is used";
              }
              type decimal64 {
                fraction-digits 2;
              }
              description
                "rate-limit traffic";
            }
          }
          container statistics {
            config false;
            description
              "Aggregate statistics.";
            uses ietf-acl:acl-counters;
          }
        }
      }
    }
  }

  container dots-data {
    description
      "Main container for DOTS data channel.";
    list dots-client {
      key "cuid";
      description
        "List of DOTS clients.";
      leaf cuid {
        type string;
        description
          "A unique identifier that is randomly generated by
           a DOTS client to prevent request collisions.";
        reference
          "RFC YYYY: Distributed Denial-of-Service Open Threat
                   Signaling (DOTS) Signal Channel Specification";
      }
      leaf cdid {
        type string;
        description
          "A client domain identifier conveyed by a
           server-domain DOTS gateway to a remote DOTS server.";
        reference
          "RFC YYYY: Distributed Denial-of-Service Open Threat
                   Signaling (DOTS) Signal Channel Specification";
      }
      container aliases {
        description
          "Set of aliases that are bound to a DOTS client.";
        uses aliases;
      }
      container acls {
        description
          "Access lists that are bound to a DOTS client.";
         uses access-lists;
      }
    }
    container capabilities {
      config false;
      description
        "Match capabilities";
      leaf-list address-family {
        type enumeration {
          enum "ipv4"  {
            description
              "IPv4 is supported.";
          }
          enum "ipv6"  {
            description
              "IPv6 is supported.";
          }
        }
        description
          "Indicates the IP address families supported by
           the DOTS server.";
```

```
         }
         leaf-list forwarding-actions {
           type identityref {
               base ietf-acl:forwarding-action;

           }
           description
             "Supported forwarding action(s).";
         }
         leaf rate-limit {
           type boolean;
           description
             "Support of rate-limit action.";
         }
         leaf-list transport-protocols {
           type uint8;
           description
             "Upper-layer protocol associated with this mapping.

              Values are taken from the IANA protocol registry:
              https://www.iana.org/assignments/protocol-numbers/
              protocol-numbers.xhtml

              For example, this field contains 6 (TCP) for a TCP
              mapping or 17 (UDP) for a UDP mapping.";
         }
         container ipv4 {
           description
             "Indicates IPv4 header fields that are supported to enforce
              ACLs.";
           leaf dscp {
             type boolean;
             description
               "Support of filtering based on DSCP.";
           }
           leaf ecn {
             type boolean;
             description
               "Support of filtering based on ECN.";
           }
           leaf length {
             type boolean;
             description
               "Support of filtering based on the Total Length.";
           }
           leaf ttl {
             type boolean;
             description
               "Support of filtering based on the TTL.";
           }
           leaf protocol {
             type boolean;
             description
               "Support of filtering based on protocol field.";
           }
           leaf ihl {
             type boolean;
             description
               "Support of filtering based on the Internet Header
                Length (IHL).";
           }
           leaf ~~source-prefix~~ flags {
             type boolean;
             description
               "Support of filtering based on the ~~source prefix."; ~~ 'flags'";
           }
           leaf ~~destination-prefix~~ offset {
             type boolean;
             description
               "Support of filtering based on the ~~destination prefix."; ~~ 'offset'.";
           }
           leaf ~~fragment~~ identification {
             type boolean;
             description
               ~~"Indicates the capability~~
               "Support of ~~a DOTS server to~~
~~              enforce filters~~ filtering based on ~~IPv4 fragments.";~~
~~         }~~ the 'identification'.";
           }
         ~~container ipv6~~
           leaf source-prefix {
           ~~description~~
             type boolean;
             description
               "Support of filtering based on the source prefix.";
           }
           leaf destination-prefix {
             type boolean;
             description
               "Support of filtering based on the destination prefix.";
           }
           leaf fragment {
             type boolean;
             description
               "Indicates the capability of a DOTS server to
                enforce filters on IPv4 fragments. That is 'fragment'
                clause is supported.";
           }
```

```
        }
        container ipv6 {
          description
            "Indicates IPv6 header fields that are supported to enforce
             ACLs.";
          leaf dscp {
            type boolean;
            description
              "Support of filtering based on DSCP.";
          }
          leaf ecn {
            type boolean;
            description
              "Support of filtering based on ECN.";
          }
          leaf flow-label {
            type boolean;
            description
              "Support of filtering based on the Flow label.";
          }
          leaf length {
            type boolean;
            description
              "Support of filtering based on the Payload Length.";
          }
          leaf protocol {
            type boolean;
            description
              "Support of filtering based on the Next Header field.";
          }
          leaf hoplimit {
            type boolean;
            description
              "Support of filtering based on the Hop Limit.";
          }
          leaf source-prefix {
            type boolean;
            description
              "Support of filtering based on the source prefix.";
          }
          leaf destination-prefix {
            type boolean;
            description
              "Support of filtering based on the destination prefix.";
          }
          leaf fragment {
            type boolean;
            description
              "Indicates the capability of a DOTS server to
               enforce filters on IPv6 fragments.";
          }
        }
        container tcp {
          description
            "Set of TCP fields that are supported by the DOTS server
             to enfoce filters.";

          leaf sequence-number {
            type boolean;
            description
              "Support of filtering based on the TCP sequence number.";
          }
          leaf acknowledgement-number {
            type boolean;
            description
              "Support of filtering based on the TCP acknowledgement
               number.";
          }
          leaf data-offset {
            type boolean;
            description
              "Support of filtering based on the TCP data-offset.";
          }
          leaf reserved {
            type boolean;
            description
              "Support of filtering based on the TCP reserved field.";
          }
          leaf flags {
            type boolean;
            description
              "Support of ~~filtering~~ filtering, as defined in RFC ZZZZ, based
               on the TCP flags.";
          }
          leaf flags-bitmask {
            type boolean;
            description
              "Support of filtering based on the TCP flags bitmask.";
          }
          leaf window-size {
            type boolean;
            description
              "Support of filtering based on the TCP window size.";
          }
          leaf urgent-pointer {
            type boolean;
            description
              "Support of filtering based on the TCP urgent pointer.";
```

```
          }
          leaf options {
            type boolean;
            description
              "Support of filtering based on the TCP options.";
          }
          leaf source-port {
            type boolean;
            description
              "Support of filtering based on the source port number.";
          }
          leaf destination-port {
            type boolean;
            description
              "Support of filtering based on the destination port
               number.";
          }
          leaf port-range {
            type boolean;
            description
              "Support of filtering based on a port range.";
          }
        }
        container udp {
          description
            "Set of UDP fields that are supported by the DOTS server
             to enforce filters.";
          leaf length {
            type boolean;
            description
              "Support of filtering based on the UDP length.";
          }
          leaf source-port {
            type boolean;
            description
              "Support of filtering based on the source port number.";
          }
          leaf destination-port {
            type boolean;
            description
              "Support of filtering based on the destination port
               number.";
          }
          leaf port-range {
            type boolean;
            description
              "Support of filtering based on a port range.";
          }
        }
        container icmp {
          description
            "Set of ICMP/ICMPv6 fields that are supported by the DOTS
             server to enforce filters.";
          leaf type {
            type boolean;
            description
              "Support of filtering based on the ICMP/ICMPv6 type.";
          }
          leaf code {
            type boolean;
            description
              "Support of filtering based on the ICMP/ICMPv6 code.";
          }
          leaf rest-of-header {
            type boolean;
            description
              "Support of filtering based on the ICMP four-bytes
               field.";
          }
        }
      }
    }
  }
   <CODE ENDS>
```

5. Managing DOTS Clients

5.1. Registering DOTS Clients

In order to make use of DOTS data channel, a DOTS client MUST register to its DOTS server(s) by creating a DOTS client ('dots-client') resource. To that aim, DOTS clients SHOULD send a POST request (shown in Figure 11).

```
POST /restconf/data/ietf-dots-data-channel:dots-data HTTP/1.1
Host: {host}:{port}
Content-Type: application/yang-data+json
{
  "ietf-dots-data-channel:dots-client": [
    {
      "cuid": "string"
    }
  ]
}
```

Figure 11: POST to Register

The 'cuid' (client unique identifier) parameter is described below:

cuid: A globally unique identifier that is meant to prevent collisions among DOTS clients. This attribute has the same meaning, syntax, and processing rules as the 'cuid' attribute defined in [I-D.ietf-dots-signal-channel].

DOTS clients MUST use the same 'cuid' for both signal and data channels.

This is a mandatory attribute.

In deployments where server-domain DOTS gateways are enabled, identity information about the origin source client domain SHOULD be supplied to the DOTS server. That information is meant to assist the DOTS server to enforce some policies. These policies can be enforced per-client, per-client domain, or both. Figure 12 shows an example of a request relayed by a server-domain DOTS gateway.

```
 POST /restconf/data/ietf-dots-data-channel:dots-data HTTP/1.1
 Host: {host}:{port}
 Content-Type: application/yang-data+json
 {
   "ietf-dots-data-channel:dots-client": [
     {
       "cuid": "string",
       "cdid": "string"
     }
   ]
 }
```

Figure 12: POST to Register (DOTS Gateway)

A server-domain DOTS gateway SHOULD add the following attribute:

cdid: This attribute has the same meaning, syntax, and processing rules as the 'cdid' attribute defined in [I-D.ietf-dots-signal-channel].

In deployments where server-domain DOTS gateways are enabled, 'cdid' does not need to be inserted when relaying DOTS methods to manage aliases (Section 6) or filtering rules (Section 7). DOTS servers are responsible for maintaining the association between 'cdid' and 'cuid' for policy enforcement purposes.

This is an optional attribute.

A request example to create a 'dots-client' resource is depicted in Figure 13. This request is relayed by a server-domain DOTS gateway as hinted by the presence of the 'cdid' attribute.

```
 POST /restconf/data/ietf-dots-data-channel:dots-data HTTP/1.1
 Host: {host}:{port}
 Content-Type: application/yang-data+json
 {
   "ietf-dots-data-channel:dots-client": [
     {
       "cuid": "dz6pHjaADkaFTbjr0JGBpw",
       "cdid": "7eeaf349529eb55ed50113"
     }
   ]
 }
```

Figure 13: POST to Register (DOTS gateway)

DOTS servers MUST limit the number of 'dots-client' resources to be created by the same DOTS client to 1 per request. Requests with multiple 'dots-client' resources MUST be rejected by DOTS servers. To that aim, the DOTS server MUST rely on the same procedure to unambiguously identify a DOTS client as discussed in Section 4.4.1 of [I-D.ietf-dots-signal-channel].

The DOTS server indicates the result of processing the POST request using status-line codes. Status codes in the range "2xx" codes are success, "4xx" codes are some sort of invalid requests and "5xx" codes are returned if the DOTS server has erred or is incapable of accepting the creation of the 'dots-client' resource. In particular,

o "201 Created" status-line is returned in the response, if the DOTS server has accepted the request.

o "400 Bad Request" status-line is returned by the DOTS server, if the request does not include a 'cuid' parameter. The error-tag "missing-attribute" is used in this case.

o "409 Conflict" status-line is returned to the requesting DOTS client, if the data resource already exists. The error-tag "resource-denied" is used in this case.

Once a DOTS client registers itself to a DOTS server, it can create/delete/retrieve aliases (Section 6) and filtering rules (Section 7).

A DOTS client MAY use the PUT request (Section 4.5 in [RFC8040]) to register a DOTS client within the DOTS server. An example is shown in Figure 14.

```
     PUT /restconf/data/ietf-dots-data-channel:dots-data\
         /dots-client=dz6pHjaADkaFTbjr0JGBpw HTTP/1.1
```

```
         Host: {host}:{port}
         Content-Type: application/yang-data+json
         {
           "ietf-dots-data-channel:dots-client": [
             {
               "cuid": "dz6pHjaADkaFTbjr0JGBpw"
             }
           ]
         }
```

Figure 14: PUT to Register

The DOTS gateway that inserted a 'cdid' in a PUT request, MUST strip the 'cdid' parameter in the corresponding response before forwarding the response to the DOTS client.

5.2. Uregistering DOTS Clients

A DOTS client de-registers from its DOTS server by deleting the 'cuid' resource. Resources bound to this DOTS client will be deleted by the DOTS server. An example of de-register request is shown in Figure 15.

```
 DELETE /restconf/data/ietf-dots-data-channel:dots-data\
        /dots-client=dz6pHjaADkaFTbjr0JGBpw HTTP/1.1
 Host: {host}:{port}
```

Figure 15: De-register a DOTS Client

6. Managing DOTS Aliases

The following sub-sections define means for a DOTS client to create aliases (Section 6.1), retrieve one or a list of aliases (Section 6.2), and delete an alias (Section 6.3).

6.1. Create Aliases

A POST or PUT request is used by a DOTS client to create aliases, for resources for which a mitigation may be requested. Such aliases may be used in subsequent DOTS signal channel exchanges to refer more efficiently to the resources under attack.

DOTS clients within the same domain can create different aliases for the same resource.

The structure of POST requests used to create aliases is shown in Figure 16.

```
 POST /restconf/data/ietf-dots-data-channel:dots-data\
      /dots-client=dz6pHjaADkaFTbjr0JGBpw HTTP/1.1
 Host: {host}:{port}
 Content-Type: application/yang-data+json
 {
  "ietf-dots-data-channel:aliases": {
    "alias": [
      {
        "name": "string",
        "target-prefix": [
          "string"
        ],
        "target-port-range": [
          {
            "lower-port": integer,
            "upper-port": integer
          }
        ],
        "target-protocol": [
          integer
        ],
        "target-fqdn": [
          "string"
        ],
        "target-uri": [
          "string"
        ]
      }
    ]
  }
}
```

Figure 16: POST to Create Aliases

The parameters are described below:

name: Name of the alias.

This is a mandatory attribute.

target-prefix: Prefixes are separated by commas. Prefixes are represented using Classless Inter-domain Routing (CIDR) notation [RFC4632]. As a reminder, the prefix length must be less than or equal to 32 (resp. 128) for IPv4 (resp. IPv6).

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

This is an optional attribute.

target-port-range: A range of port numbers.

The 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], the range of port numbers can be, for example, 1024-65535.

This is an optional attribute.

target-protocol: A list of protocols. Values are taken from the IANA protocol registry [proto_numbers].

The value '0' has a special meaning for 'all protocols'.

This is an optional attribute.

target-fqdn: A list of Fully Qualified Domain Names (FQDNs). An FQDN is the full name of a resource, rather than just its hostname. For example, "venera" is a hostname, and "venera.isi.edu" is an FQDN [RFC1983].

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'.

This is an optional attribute.

target-uri: A list of Uniform Resource Identifiers (URIs) [RFC3986].

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

This is an optional attribute.

In POST or PUT requests, at least one of the 'target-prefix', 'target-fqdn', or 'target-uri' attributes MUST be present. DOTS agents can safely ignore Vendor-Specific parameters they don't understand.

Figure 17 shows a POST request to create an alias called "https1" for HTTPS servers with IP addresses 2001:db8:6401::1 and 2001:db8:6401::2 listening on port number 443.

```
POST /restconf/data/ietf-dots-data-channel:dots-data\
     /dots-client=dz6pHjaADkaFTbjr0JGBpw HTTP/1.1
Host: www.example.com
Content-Type: application/yang-data+json
{
  "ietf-dots-data-channel:aliases": {
    "alias": [
      {
        "name": "https1",
        "target-protocol": [
          6
        ],
        "target-prefix": [
          "2001:db8:6401::1/128",
          "2001:db8:6401::2/128"
        ],
        "target-port-range": [
          {
            "lower-port": 443
          }
        ]
      }
    ]
  }
}
```

Figure 17: Example of a POST to Create an Alias

"201 Created" status-line MUST be returned in the response if the DOTS server has accepted the alias.

"409 Conflict" status-line MUST be returned to the requesting DOTS client, if the request is conflicting with an existing alias name. The error-tag "resource-denied" is used in this case.

If the request is missing a mandatory attribute or its contains an invalid or unknown parameter, "400 Bad Request" status-line MUST be returned by the DOTS server. The error-tag is set to "missing-attribute", "invalid-value", or "unknown-element" as a function of the encountered error.

If the request is received via a server-domain DOTS gateway, but the DOTS server does not maintain a 'cdid' for this 'cuid' while a 'cdid' is expected to be supplied, the DOTS server MUST reply with "403 Forbidden" status-line and the error-tag "access-denied". Upon receipt of this message, the DOTS client MUST register (Section 5).

A DOTS client uses the PUT request to modify the aliases in the DOTS server. In particular, a DOTS client MUST update its alias entries upon change of the prefix indicated in the 'target-prefix'.

A DOTS server MUST maintain an alias for at least 10080 minutes (1 week). If no refresh request is seen from the DOTS client, the DOTS server removes expired entries.

6.2. Retrieve Installed Aliases

GET request is used to retrieve one or all installed aliases by a DOTS client from a DOTS server (Section 3.3.1 in [RFC8040]). If no 'name' is included in the request, this is an indication that the request is about retrieving all aliases instantiated by the DOTS client.

Figure 18 shows an example to retrieve all the aliases that were instantiated by the requesting DOTS client. The 'content' parameter and its permitted values are defined in Section 4.8.1 of [RFC8040].

```
  GET /restconf/data/ietf-dots-data-channel:dots-data\
      /dots-client=dz6pHjaADkaFTbjr0JGBpw\
      /aliases?content=all HTTP/1.1
  Host: {host}:{port}
  Accept: application/yang-data+json
```

Figure 18: GET to Retrieve All Installed Aliases

Figure 19 shows an example of the response message body that includes all the aliases that are maintained by the DOTS server for the DOTS client identified by the 'cuid' parameter.


```
{
  "ietf-dots-data-channel:aliases": {
    "alias": [
      {
        "name": "Server1",
        "target-protocol": [
          6
        ],
        "target-prefix": [
          "2001:db8:6401::1/128",
          "2001:db8:6401::2/128"
        ],
        "target-port-range": [
          {
            "lower-port": 443
          }
        ],
        "pending-lifetime": 3596
      },
      {
        "name": "Server2",
        "target-protocol": [
          6
        ],
        "target-prefix": [
          "2001:db8:6401::10/128",
          "2001:db8:6401::20/128"
        ],
        "target-port-range": [
          {
            "lower-port": 80
          }
        ],
        "pending-lifetime": 9869
      }
    ]
  }
}
```


Figure 19: An Example of Response Body

Figure 20 shows an example of a GET request to retrieve the alias "Server2" that was instantiated by the DOTS client.

```
  GET /restconf/data/ietf-dots-data-channel:dots-data\
      /dots-client=dz6pHjaADkaFTbjr0JGBpw\
      /aliases/alias=Server2?content=all HTTP/1.1
  Host: {host}:{port}
  Accept: application/yang-data+json
```

Figure 20: GET to Retrieve an Alias

If an alias name ('name') is included in the request, but the DOTS server does not find that alias name for this DOTS client in its configuration data, it MUST respond with a "404 Not Found" status-line.

6.3. Delete Aliases

DELETE request is used to delete an alias maintained by a DOTS server.

If the DOTS server does not find the alias name, conveyed in the DELETE request, in its configuration data for this DOTS client, it MUST respond with a "404 Not Found" status-line.

The DOTS server successfully acknowledges a DOTS client's request to remove the alias using "204 No Content" status-line in the response.

Figure 21 shows an example of a request to delete an alias.

```
DELETE /restconf/data/ietf-dots-data-channel:dots-data\
       /dots-client=dz6pHjaADkaFTbjr0JGBpw\
       /aliases/alias=Server1 HTTP/1.1
Host: {host}:{port}
```

Figure 21: Delete an Alias

7. Managing DOTS Filtering Rules

The following sub-sections define means for a DOTS client to retrieve DOTS filtering capabilities (Section 7.1), create filtering rules (Section 7.2), retrieve active filtering rules (Section 7.3), and delete a filtering rule (Section 7.4).

7.1. Retrieve DOTS Filtering Capabilities

A DOTS client MAY send a GET request to retrieve the filtering capabilities supported by a DOTS server. Figure 22 shows an example of such request.

```
GET /restconf/data/ietf-dots-data-channel:dots-data\
    /capabilities HTTP/1.1
Host: {host}:{port}
Accept: application/yang-data+json
```

Figure 22: GET to Retrieve the Capabilities of a DOTS Server

A DOTS client which issued a GET request to retrieve the filtering capabilities supported by its DOTS server, SHOULD NOT request for filtering actions that are not supported by that DOTS server.

Figure 23 shows an example of a response received from a DOTS server which only supports the mandatory filtering criteria listed in Section 4.1.

```
 Content-Type: application/yang-data+json
 {
  "ietf-dots-data-channel:capabilities": {
    "address-family": ["ipv4", "ipv6"],
    "forwarding-actions": ["drop", "accept"],
    "rate-limit": true,
    "transport-protocols": [1, 6, 17, 58],
    "ipv4": {
      "length": true,
      "protocol": true,
      "destination-prefix": true,
      "source-prefix": true,
      "fragment": true
    },
    "ipv6": {
      "length": true,
      "protocol": true,
      "destination-prefix": true,
      "source-prefix": true,
      "fragment": true
    },
    "tcp": {
      ~~"flags":~~
      "flags-bitmask": true,
      "source-port": true,
      "destination-port": true,
      "port-range": true
    },
    "udp": {
      "length": true,
      "source-port": true,
      "destination-port": true,
      "port-range": true
    },
    "icmp": {
      "type": true,
      "code": true
    }
  }
}
```

Figure 23: Reply to a GET Response with Filtering Capabilities

7.2. Install Filtering Rules

A POST or PUT request is used by a DOTS client to communicate filtering rules to a DOTS server.

Figure 24 shows a POST request example to block traffic from 192.0.2.0/24 and destined to 198.51.100.0/24. Other examples are discussed in Appendix A.

```
 POST /restconf/data/ietf-dots-data-channel:dots-data\
      /dots-client=dz6pHjaADkaFTbjr0JGBpw HTTP/1.1
```

```
 Host: {host}:{port}
 Content-Type: application/yang-data+json
 {
  "ietf-dots-data-channel:acls": {
    "acl": [
      {
        "name": "sample-ipv4-acl",
        "type": "ipv4-acl-type",
        "activation-type": "activate-when-mitigating",
        "aces": {
          "ace": [
            {
              "name": "rule1",
              "matches": {
                "ipv4": {
                  "destination-ipv4-network": "198.51.100.0/24",
                  "source-ipv4-network": "192.0.2.0/24"
                }
              },
              "actions": {
                "forwarding": "drop"
              }
            }
          ]
        }
      }
    ]
  }
 }
```

Figure 24: POST to Install Filtering Rules

The meaning of these parameters is as follows:

name: The name of the access list.

This is a mandatory attribute.

type: Indicates the primary intended type of match criteria (e.g., IPv4, IPv6). It is set to 'ipv4-acl-type' in this example.

This is an optional attribute.

activation-type: Indicates whether an ACL has to be installed immediately or during mitigation time. If this attribute is not provided, the DOTS server MUST use 'activate-when-mitigating' as default value. Filters that are activated only when a mitigation is in progress MUST be bound to the DOTS client which created the filtering rule.

This is an optional attribute.

matches: Define criteria used to identify a flow on which to apply the rule. It can be "l3" (IPv4, IPv6) or "l4" (TCP, UDP, ..). The detailed match parameters are specified in Section 4.

In this example, an IPv4 matching criteria is used.

This is an optional attribute.

destination-ipv4-network: The destination IPv4 prefix. DOTS servers MUST validate that these prefixes are within the scope of the DOTS client's domain. Other validation checks may be supported by DOTS servers. If this attribute is not provided, the DOTS server enforces the ACL on any destination IP address that belong to the DOTS client's domain.

This is a mandatory attribute in requests with an 'activation-type' set to 'immediate'.

source-ipv4-network: The source IPv4 prefix.

This is an optional attribute.

actions: Actions in the forwarding ACL category can be "drop" or "accept". The "accept" action is used to white-list traffic. The "drop" action is used to black-list traffic.

Accepted traffic may be subject to "rate-limit"; the allowed traffic rate is represented in bytes per second indicated in IEEE floating point format [IEEE.754.1985].

This is a mandatory attribute.

The DOTS server indicates the result of processing the POST request using the status-line header. Concretely, "201 Created" status-line MUST be returned in the response if the DOTS server has accepted the filtering rules. If the request is missing a mandatory attribute or contains an invalid or unknown parameter (e.g., a match field not supported by the DOTS server), "400 Bad Request" status-line MUST be returned by the DOTS server in the response. The error-tag is set to "missing-attribute", "invalid-value", or "unknown-element" as a function of the encountered error.

If the request is received via a server-domain DOTS gateway, but the DOTS server does not maintain a 'cdid' for this 'cuid' while a 'cdid' is expected to be supplied, the DOTS server MUST reply with "403 Forbidden" status-line and the error-tag "access-denied". Upon receipt of this message, the DOTS client MUST register (Figure 11).

If the request is conflicting with an existing filtering installed by another DOTS client of the domain, the DOTS server returns "409 Conflict" status-line to the requesting DOTS client. The error-tag "resource-denied" is used in this case.

The "insert" query parameter (Section 4.8.5 of [RFC8040]) MAY be used to specify how an access control entry is inserted within an ACL and how an ACL is inserted within an ACL set.

The DOTS client uses the PUT request to modify its filtering rules maintained by the DOTS server. In particular, a DOTS client MUST update its filtering entries upon change of the destination-prefix. How such change is detected is out of scope.

A DOTS server MUST maintain a filtering rule for at least 10080 minutes (1 week). If no refresh request is seen from the DOTS client, the DOTS server removes expired entries. Typically, a refresh request is a PUT request which echoes the content of a response to a GET request with all of the read-only parameters stripped out (e.g. pending-lifetime).

7.3. Retrieve Installed Filtering Rules

The DOTS client periodically queries the DOTS server to check the counters for installed filtering rules. GET request is used to retrieve filtering rules from a DOTS server. **In order to indicate which type of data is requested in a GET request, the DOTS client sets adequately the 'content' parameter.**

If the DOTS server does not find the access list name conveyed in the GET request in its configuration data for this DOTS client, it responds with a "404 Not Found" status-line.

In order to illustrate the intended behavior, consider the example depicted in Figure ~~25 shows how~~ **25. In reference** to ~~retrieve all~~ **this example,** the ~~filtering rules~~ **DOTS client requests the creation of an immediate ACL called "test-acl-ipv6-udp".**

```
PUT /restconf/data/ietf-dots-data-channel:dots-data\
    /dots-client=paL8p4Zqo4SLv64TLPXrxA/acls\
    /acl=test-acl-ipv6-udp HTTP/1.1
Host: {host}:{port}
Content-Type: application/yang-data+json
{
  "ietf-dots-data-channel:acls": {
    "acl": [
      {
        "name": "test-acl-ipv6-udp",
        "type": "ipv6-acl-type",
        "activation-type": "immediate",
        "aces": {
          "ace": [
            {
              "name": "test-ace-ipv6-udp",
              "matches": {
                "ipv6": {
                  "destination-ipv6-network": "2001:db8:6401::2/127",
                  "source-ipv6-network": "2001:db8:1234::/96",
                  "protocol": 17,
                  "flow-label": 10000
                },
                "udp": {
                  "source-port": {
                    "operator": "lte",
                    "port": 80
                  },
                  "destination-port": {
                    "operator": "neq",
                    "port": 1010
                  }
                }
              },
              "actions": {
                "forwarding": "accept"
              }
            }
          ]
        }
      }
    ]
  }
}
```

Figure 25: Example of a PUT Request to Create a Filtering

The peer DOTS server follows the procedure specified in Section 7.2 to process the request. We consider in the following that ~~were instantiated~~ **a positive response is sent back to the requesting DOTS client to confirm that the ""test-acl-ipv6-udp" ACL is successfully installed** by the DOTS **server.**

The DOTS client **can issue a GET request to retrieve all its filtering rules** and the number of matches for the installed filtering ~~rules.~~ **rules as illustrated in Figure 26. The message body of the response to this GET request is shown in Figure 27.**

```
  GET /restconf/data/ietf-dots-data-channel:dots-data\
      /dots-client=dz6pHjaADkaFTbjr0JGBpw\
      /acls?content=all HTTP/1.1
  Host: {host}:{port}
  Accept: application/yang-data+json
```

Figure ~~25: GET to~~ **26:** Retrieve the Configuration Data and State Data for the Filtering ~~Rules~~ **Rules: GET Request**


```
{
  "ietf-dots-data-channel:acls": {
    "acl": [
      {
        "name": "test-acl-ipv6-udp",
        "type": "ipv6-acl-type",
        "activation-type": "immediate",
        "pending-lifetime":9080,
        "aces": {
          "ace": [
            {
              "name": "test-ace-ipv6-udp",
              "matches": {
                "ipv6": {
                  "destination-ipv6-network": "2001:db8:6401::2/127",
                  "source-ipv6-network": "2001:db8:1234::/96",
                  "protocol": 17,
                  "flow-label": 10000
                },
                "udp": {
                  "source-port": {
                    "operator": "lte",
                    "port": 80
                  },
                  "destination-port": {
                    "operator": "neq",
                    "port": 1010
                  }
                }
              },
              "actions": {
                "forwarding": "accept"
              }
            }
          ]
        }
      }
    ]
  }
}
```


Figure ~~26 shows how to retrieve "sample-ipv6-acl" filtering rule instantiated by~~ **27: Retrieve** the ~~DOTS client, having "cuid=dz6pHjaADkaFTbjr0JGBpw",~~ **Configuration Data** and **State Data for** the ~~number of matches~~ **Filtering Rules: Response**

Also, a DOTS client can issue a GET request to retrieve only configuration data related to an ACL as shown in Figure 28.

```
  GET /restconf/data/ietf-dots-data-channel:dots-data\
      /dots-client=paL8p4Zqo4SLv64TLPXrxA/acls\
      /acl=test-acl-ipv6-udp?content=config HTTP/1.1
  Host: {host}:{port}
  Accept: application/yang-data+json
```

Figure 28: Retrieve the Configuration Data for **a Filtering Rule: GET Request**

A response to this GET request is shown in Figure 29.


```
{
  "ietf-dots-data-channel:acls": {
    "acl": [
      {
        "name": "test-acl-ipv6-udp",
        "type": "ipv6-acl-type",
        "activation-type": "immediate",
        "aces": {
          "ace": [
            {
              "name": "test-ace-ipv6-udp",
              "matches": {
                "ipv6": {
                  "destination-ipv6-network": "2001:db8:6401::2/127",
                  "source-ipv6-network": "2001:db8:1234::/96",
                  "protocol": 17,
                  "flow-label": 10000
                },
                "udp": {
                  "source-port": {
                    "operator": "lte",
                    "port": 80
                  },
                  "destination-port": {
                    "operator": "neq",
                    "port": 1010
                  }
                }
              },
```

```
                "actions": {
                  "forwarding": "accept"
                }
              }
            ]
          }
        }
      ]
    }
  }
```

Figure 29: Retrieve the ~~installed filtering rules.~~ Configuration Data for a Filtering Rule: Response

A DOTS client can also issue a GET request to retrieve only non-configuration data bound to a given ACL as shown in Figure 28. A response to this GET request is shown in Figure 31.

```
  GET /restconf/data/ietf-dots-data-channel:dots-data\
      ~~/dots-client=dz6pHjaADkaFTbjr0JGBpw/acls\~~
      ~~/acl=sample-ipv6-acl?content=all~~
      /dots-client=paL8p4Zqo4SLv64TLPXrxA/acls\
      /acl=test-acl-ipv6-udp?content=non-config HTTP/1.1
  Host: {host}:{port}
  Accept: application/yang-data+json
```

Figure ~~26: GET to~~ 30: Retrieve the ~~Configuration~~ Non-Configuration Data ~~and State~~ for a Filtering Rule: GET Request


```
{
  "ietf-dots-data-channel:acls":  {
    "acl": [
      {
        "name": "test-acl-ipv6-udp",
        "pending-lifetime": 8000,
        "aces": {
          "ace": [
            {
              "name": "test-ace-ipv6-udp"
            }
          ]
        }
      }
    ]
  }
}
```


Figure 31: Retrieve the Non-Configuration Data for a Filtering ~~Rule~~ Rule: GET Request

7.4. Remove Filtering Rules

DELETE request is used by a DOTS client to delete filtering rules from a DOTS server.

If the DOTS server does not find the access list name carried in the DELETE request in its configuration data for this DOTS client, it MUST respond with a "404 Not Found" status-line. The DOTS server successfully acknowledges a DOTS client's request to withdraw the filtering rules using "204 No Content" status-line, and removes the filtering rules accordingly.

Figure ~~27~~ 32 shows an example of a request to remove the IPv4 ACL ~~named "sample-ipv4-acl".~~ "sample-ipv4-acl" created in Section 7.2.

```
  DELETE  /restconf/data/ietf-dots-data-channel:dots-data\
          /dots-client=dz6pHjaADkaFTbjr0JGBpw/acls\
          /acl=sample-ipv4-acl HTTP/1.1
  Host: {host}:{port}
```

Figure ~~27:~~ 32: Remove a Filtering Rule: DELETE Request

Figure 33 shows an example of a response received from the server to confirm the deletion of "sample-ipv4-acl".

```
 HTTP/1.1 204 No Content
 Server: Apache
 Date: Fri, 27 Jul 2018 10:05:15 GMT
 Cache-Control: no-cache
 Content-Type: application/yang-data+json
 Content-Length: 0
 Connection: Keep-Alive
```

Figure 33: Remove a Filtering ~~Rule~~ Rule: Response

8. IANA Considerations

This document requests IANA to register the following URI in the "IETF XML Registry" [RFC3688]:

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

This document requests IANA to register the following YANG module in the "YANG Module Names" registry [RFC7950].

```
name: ietf-dots-data-channel
namespace: urn:ietf:params:xml:ns:yang:ietf-dots-data-channel
prefix: data-channel
reference: RFC XXXX
```

9. ~~Contributors~~

~~The following individuals have contributed to this document:~~

~~o Dan Wing, Email: dwing-ietf@fuggles.com~~

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

~~10.~~ Security Considerations

RESTCONF security considerations are discussed in [RFC8040]. In particular, DOTS agents MUST follow the security recommendations in Sections 2 and 12 of [RFC8040]. Also, DOTS agents MUST support the mutual authentication TLS profile discussed in Sections 7.1 and 8 of [I-D.ietf-dots-signal-channel]. YANG ACL-specific security considerations are discussed in [I-D.ietf-netmod-acl-model].

Authenticated encryption MUST be used for data confidentiality and message integrity. The interaction between the DOTS agents requires Transport Layer Security (TLS) with a cipher suite offering confidentiality protection and the guidance given in [RFC7525] MUST be followed to avoid attacks on TLS.

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-AO [RFC5925]. 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.

In order to prevent leaking internal information outside a client-domain, client-side DOTS gateways SHOULD NOT reveal the identity of 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 enforce filtering rules on a given IP prefix. That is, only filtering rules on IP resources that belong to the DOTS client's domain MUST be authorized by a DOTS server. The exact mechanism for the DOTS servers to validate that the target prefixes are within the scope of the DOTS client's domain is deployment-specific.

Rate-limiting DOTS requests, including those with new 'cuid' values, from the same DOTS client defends 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.

Applying resources quota per DOTS client and/or per DOTS client domain (e.g., limit the number of aliases and filters to be install by DOTS clients) prevents DOTS server resources to be aggressively used by some DOTS clients and ensures, therefore, DDoS mitigation usage fairness. Additionally, DOTS servers may limit the number of DOTS clients that can be enabled per domain.

The presence of DOTS gateways may lead to infinite forwarding loops, which is undesirable. To prevent and detect such loops, a mechanism is defined in Section 3.5.

All data nodes defined in the YANG module which can be created, modified, and deleted (i.e., config true, which is the default) are considered sensitive. Write operations applied to these data nodes without proper protection can negatively affect network operations. Appropriate security measures are recommended to prevent illegitimate users from invoking DOTS data channel primitives. Nevertheless, an attacker who can access a DOTS client is technically capable of launching various attacks, such as:

- Set an arbitrarily low rate-limit, which may prevent legitimate traffic from being forwarded (rate-limit).
- Set an arbitrarily high rate-limit, which may lead to the forwarding of illegitimate DDoS traffic (rate-limit).
- Communicate invalid aliases to the server (alias), which will cause the failure of associating both data and signal channels.
- Set invalid ACL entries, which may prevent legitimate traffic from being forwarded. Likewise, invalid ACL entries may lead to forward DDoS traffic.

10. Contributors

The following individuals have contributed to this document:

- **Dan Wing, Email: dwing-ietf@fuggles.com**
- **Jon Shallow, NCC Group, Email: jon.shallow@nccgroup.trust**

11. Acknowledgements

Thanks to Christian Jacquenet, Roland Dobbins, Roman Danyliw, Ehud Doron, Russ White, Gilbert Clark, and Nesredien Suleiman for the discussion and comments.

12. References

12.1. Normative References

[I-D.ietf-dots-signal-channel]
Reddy, T., Boucadair, M., Patil, P., Mortensen, A., and N. Teague, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel Specification", draft- ~~ietf-dots-signal-channel-20~~ **ietf-dots-signal-channel-21** (work in progress), ~~May~~ **July** 2018.

[I-D.ietf-netmod-acl-model]
Jethanandani, M., Huang, L., Agarwal, S., and D. Blair, "Network Access Control List (ACL) YANG Data Model", draft-ietf-netmod-acl-model-19 (work in progress), April 2018.

[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>.

[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>.

[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>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[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>.

[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>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

12.2. Informative References

[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-06 (work in progress), March 2018.

[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-14 (work in progress), February 2018.

[IEEE.754.1985]
Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", August 1985.

[proto_numbers]
"IANA, "Protocol Numbers"", 2011, <http://www.iana.org/assignments/protocol-numbers>.

[RFC1983] Malkin, G., Ed., "Internet Users' Glossary", FYI 18, RFC 1983, DOI 10.17487/RFC1983, August 1996, <https://www.rfc-editor.org/info/rfc1983>.

[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>.

[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>.

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

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <https://www.rfc-editor.org/info/rfc5389>.

[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>.

[RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, DOI 10.17487/RFC6520, February 2012, <https://www.rfc-editor.org/info/rfc6520>.

[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>.

[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>.

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

[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>.

Appendix A. Sample Examples: Filtering Fragments

This specification strongly recommends the use of "fragment" for handling fragments.

Figure ~~28~~ **34** shows ~~a~~ **the content of the** POST request ~~example~~ **to be** issued by a DOTS client to its DOTS server to allow the traffic destined to 198.51.100.0/24 and UDP port number 53, but to drop all fragmented packets. The following ACEs are defined (in this order):

- ~~"drop-all-except-last-fragment"~~ **"drop-all-fragments"** ACE: discards all ~~fragments, except the last fragment.~~ **fragments (including atomic fragments).**

- "allow-dns-packets" ACE: accepts DNS packets destined to 198.51.100.0/24.

- ~~"drop-last-fragment" ACE: drops the last fragment.~~

~~The ACEs order is important to appropriately enforce the intended filtering policy. For example, if the ACEs order is "allow-dns-packets" ACE, "drop-all-except-last-fragment" ACE, and then "drop-last-fragment" ACE, the first fragment won't be dropped because it includes both L3 and L4 information and will therefore match the "allow-dns-packets" ACE.~~

```
 POST /restconf/data/ietf-dots-data-channel:dots-data\
      /dots-client=dz6pHjaADkaFTbjr0JGBpw HTTP/1.1
 Host: {host}:{port}
 Content-Type: application/yang-data+json
 {
  "ietf-dots-data-channel:acls": {
    "acl": [
      {
        "name": "dns-fragments",
        "type": "ipv4-acl-type",
        "aces": {
          "ace": [
            {
              "name": "drop-all-except-last-fragment", "drop-all-fragments",
              "matches": {
                "ipv4": {
                  "flags": "more"
                  "fragment": {
                    "operator": "match",
                    "type": "isf"
                  }
                }
              },
              "actions": {
                "forwarding": "drop"
              }
            }
          ]
          "ace": [
            {
              "name": "allow-dns-packets",
              "matches": {
                "ipv4": {
                  "destination-ipv4-network": "198.51.100.0/24"
                }
```

```
                "udp": {
                  "destination-port": {
                    "operator": "eq",
                    "port": 53
                  }
                },
                "actions": {
                  "forwarding": "accept"
                }
              }
            ]
            ~~"ace": [~~
              ~~{~~
                ~~"name": "drop-last-fragment",~~
                ~~"matches": {~~
                  ~~"ipv4": {~~
                    ~~"flags": ""~~
                  ~~}~~
                ~~},~~
                ~~"actions": {~~
                  ~~"forwarding": "drop"~~
                ~~}~~
              ~~}~~
            ~~]~~
          }
        }
      ]
    }
   }
```

Figure ~~28:~~ **34:** Filtering IPv4 Fragmented Packets **(Recommended)**

Figure ~~29~~ **35** shows a POST request example issued by a DOTS client to its DOTS server to allow the traffic destined to 2001:db8::/32 and UDP port number 53, but to drop all fragmented packets. The following ACEs are defined (in this order):

o "drop-all-fragments" ACE: discards all fragments (including atomic fragments). That is, IPv6 packets which include a Fragment header (44) are dropped.

o "allow-dns-packets" ACE: accepts DNS packets destined to 2001:db8::/32.

```
 POST /restconf/data/ietf-dots-data-channel:dots-data\
      /dots-client=dz6pHjaADkaFTbjr0JGBpw HTTP/1.1
 Host: {host}:{port}
 Content-Type: application/yang-data+json
 {
  "ietf-dots-data-channel:acls": {
    "acl": [
      {
        "name": "dns-fragments",
        "type": "ipv6-acl-type",
        "aces": {
          "ace": [
            {
              "name": "drop-all-fragments",
              "matches": {
                "ipv6": {
                  "fragment": ~~[null]~~ {
                    "operator": "match",
                    "type": "isf"
                  }
                }
              },
              "actions": {
                "forwarding": "drop"
              }
            }
          ]
          "ace": [
            {
              "name": "allow-dns-packets",
              "matches": {
                "ipv6": {
                  "destination-ipv6-network": "2001:db8::/32"
                }
                "udp": {
                  "destination-port": {
                    "operator": "eq",
                    "port": 53
                  }
                }
              },
              "actions": {
                "forwarding": "accept"
              }
            }

          ]
        }
      }
    ]
  }
 }
```

Figure ~~29:~~ **35:** Filtering IPv6 Fragmented Packets

Authors' Addresses

Mohamed Boucadair (editor)
Orange
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Tirumaleswar Reddy (editor)
McAfee, Inc.
Embassy Golf Link Business Park
Bangalore, Karnataka 560071
India

Email: kondtir@gmail.com

~~Mohamed Boucadair (editor)~~
~~Orange~~
~~Rennes 35000~~
~~France~~

~~Email: mohamed.boucadair@orange.com~~

Kaname Nishizuka
NTT Communications
GranPark 16F 3-4-1 Shibaura, Minato-ku
Tokyo 108-8118
Japan

Email: kaname@nttv6.jp

Liang Xia
Huawei
101 Software Avenue, Yuhuatai District
Nanjing, Jiangsu 210012
China

Email: frank.xialiang@huawei.com
Prashanth Patil
Cisco Systems, Inc.

Email: praspati@cisco.com

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

Email: amortensen@arbor.net

Nik Teague
Verisign, Inc.
United States

Email: nteague@verisign.com