DOTS K. Nishizuka
Internet-Draft NTT Communications
Intended status: Standards Track L. Xia
Expires: January 2, 2017 J. Xia
Huawei Technologies Co., Ltd.
D. Zhang

L. Fang
Microsoft
C. Gray
Comcast, Inc.
R. Compton
Charter Communications, Inc.
July 1, 2016

Inter-domain cooperative DDoS protection mechanism
draft-nishizuka-dots-inter-domain-mechanism-01

Abstract

As DDoS attacks evolve rapidly in the aspect of volume and sophistication, cooperation among operators becomes very necessary because it will give us quicker and more sophisticated protection to cope with them. This document describes possible mechanisms which implement the cooperative inter-domain DDoS protection by DOTS protocol. The described data models are intended to cover intra-domain and inter-domain solutions.

Copyright Notice

Table of Contents

1. Introduction

These days, DDoS attacks are getting bigger and more sophisticated. Preliminary measures for minimizing damages caused by such attacks are indispensable to all organizations facing to the Internet. Due to the various types of UDP reflection attacks that can be executed, there are still large DDoS attacks being generated which consist of vulnerable servers, broadband routers and other network equipment distributed all over the world. Because of the amplification feature of the reflection attack, attackers can generate massive attacks with small resources. Moreover, there are many booters who are selling DDoS attacks as a service. DDoS attacks are commoditized, so the frequency of DDoS attacks are also increasing.

These trends of attacks could exceed the capacity of a protection system of one organization in the aspect of volume and frequency. Therefore, sharing the capacity and capability of protection systems with each other to cope with such attacks becomes very necessary.

By utilizing other organization's resources, the burden of the protection is shared. The shared resources are not only CPU/memory resources of dedicated mitigation devices but also the capability of mitigation actions such as blackholing and filtering. We call the protections which utilize shared resources "cooperative DDoS protection".

Cooperative DDoS protection has numerous merits. First, as described above, it can leverage expanded capacity of protection by sharing the resources among organizations. Generally DDoS attacks happen unexpectedly, thus the capacity utilization ratio of a protection system is not constant. So, while the utilization ratio is low, it can be used by another organization which is under attack. Second, organizations can implement various countermeasures. If an attack is highly sophisticated and there is no countermeasure in the victim's system, cooperative DDoS protection can offer an optimal countermeasure for all partners. Third, it can block malicious traffic nearer to the origin of the attack. Near source defense is ideal for the health of the Internet because it can reduce the total cost of forwarding packets which, in the case of DDoS attacks mostly consist of useless massive attack traffic. Moreover, it is also very effective to solve the inter-domain uplink congestion problem. Finally, it can reduce the time to respond to an attack. After getting attacked, prompt response is important because outage of service can cause significant loss to the victim organization. This cooperating channel between partner organizations is automated by DOTS protocol.

The proposed solutions are covering both intra-domain and inter-domain situations. This standardized approach can utilize various protection systems in automated manner which can afford it quicker and more sophisticated protection in its domain.

1.1. Scope

The solutions described in this draft are based on intra-domain and inter-domain usecases in [I-D.draft-ietf-dots-use-cases]. The DOTS protocols coordinating DDoS protection in inter-domain situations in this draft are compliant with requirements in [I-D.draft-ietf-dots-requirements]. Generally DOTS is assumed to be most effective when aiding coordination of attack response between two or more organizations, but single domain scenarios are also valuable[I-D.draft-mortensen-dots-architecture]. The data model described in this draft is mainly focusing on inter-domain coordination of DDoS protection because it also covers single domain scenarios. The information required in single domain scenarios is assumed to be a subset of the information required in inter-domain scenarios.

2. Terminology

2.1. Key Words

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].

2.2. Definition of Terms

This document uses the terms defined in [I-D.draft-ietf-dots-requirements].

3. Cooperative DDoS Protection Requirements

In this section, problems regarding cooperative DDoS protection are described.

3.1. Provisioning Requirements

In inter-domain situation, a DOTS client is in a different organization from a DOTS server. To enable the protection in other organization, provisioning information should be informed to a DOTS server in advance. In the later section, the total scenario is divided into two stages: provisioning stage and signaling stage. In provisioning stage, a DOTS client is required to communicate registration messages with DOTS server which include the capacity building of protection. The data model of registration message is

defined in the protocol section of this draft. It is also required to find a way to provision other organization's DDoS protection service in secure manner. All of the messages should have confidentiality, integrity and authenticity. The requirments of the message protocol is following [I-D.draft-ietf-dots-requirements].

3.1.1. Automatic Provisioning vs Manual Provisioning

Manual provisioning is an easier way to utilize DDoS protection service of other organizations. An organization can establish trust with another organization that is going to use its DDoS protection service by many different means such as phone, e-mail, Web portal, etc,. However, it will take much time to manually provision the DDoS protection system. Attacks that occur before the DDoS system is provisioned make significant impact on the protected service. To reduce the time to start the protection, automatic provisioning is desirable. If an organization could acquire relevant information of the DDoS protection service of other organization and utilize it by DOTS signaling in a short time, the cooperative DDoS protection will succeed at a certain level. Other important work carried out in the bootstrapping process are auto-discovery and automatic capability building between the member DDoS protection service providers as the basis for the following coordination process.

3.2. Coordination Requirements

The number of the member DDoS protection service providers that will be providing cooperative DDoS protection is an important factor. If only two providers are involved, there is a bilateral relationship only. It is easy to negotiate the capacity of their own DDoS protection system. In a state of emergency, one can decide to ask for help from the other if the capacity of its own system is insufficient. When a lot of providers are joining cooperative DDoS protection, it is difficult to decide where to ask for help. They need to negotiate the capacity with every participant. It is needed to take into account all combinations to do appropriate protection. The coordination between the member providers cooperative DDoS protection is a complete process consisting of mitigation start/stop, status notification, mitigation policy updates and so on. The Inter-domain DOTS architectures described in the later section are intended to fulfill these requirements.

In addition, inter-domain uplink congestion problems can only be solved by coordinating protection services provided by the upstream operators.

3.2.1. Near Source Protection Problem

Stopping malicious traffic at the nearest point on the Internet will reduce the exhaustion of resources in all paths of the attack. To find the entry point of the attack, traceback of the attack traffic to its origin is needed. If there is a cooperative partner near the attack source, asking for help blocking the malicious traffic from the ISP is most effective.

However, the problem is that it is difficult to decide which ISP is nearest to the attack source because in many cases source addresses of attack packets are spoofed to prevent the true source from being discovered. Moreover, some topology information of an ISP's network will need to be uncovered in order to make a correct decision, however there could be privacy protection issues between ISPs. These problems can lead to difficulties locating the true attack source. These problems can be divided into two issues. The first is how to find the attacker. The second is how to decide who to ask for help.

3.3. Returning Path Requirements

As one of protection methods, some DDoS protection service provider announce BGP route to detour the attack traffic to their own network to deal with it. After scrubbing, cleaned traffic is returned to the original destination. The returning path is often called a "clean pipe". The DDoS service provider should be careful about causing routing loops because if the end point of a clean pipe is still included in the reach of the announced BGP route, the traffic will return to the mitigation path again and again. When thinking about cooperative DDoS protection, returning path information should be propagated to partners.

4. Inter-domain DOTS Architecture

With the fast growth of DDoS attack volume and sophistication, a global cooperative DDoS protection service is desirable. This service can not only address the inter-domain uplink congestion problem, but also take full advantage of global DDoS mitigation resources from different operators efficiently and enable mitigation near the source of the attack. Moreover, with providing DDoS mitigation as service, more customers will get the service flexibly they demand with maximized territory and resources. Together with on-premise DDoS protection appliances, the multiple layer DDoS protection system provides a comprehensive DDoS protection against all types of attacks, such as application layer attacks, network layer large traffic attacks and others.

The DOTS protocol is used among DOTS agents to facilitate the coordinated DDoS protection service as a whole. [I-D.draft-ietf-dots-use-cases] lists most options that DOTS agents could be used for, and describes their communication. Although this document is initiated to specify the DOTS protocol for inter-domain use cases, the final protocol would and should be the same since it is all about the signaling messages and their process between the DOTS clients and DOTS servers essentially. In other words, the protocol described here would also apply to all the intra-domain use cases. To support all the identified use cases and possibly new use cases in the future, the DOTS protocol must be extensible in terms of the message definition, protocol process, etc., which will be discussed in detail in the following section. The text below discusses the protocol mainly in respect to inter-domain use cases.

The inter-domain DDoS protection service is set up by the member operators' own DDoS protection systems and the coordination protocol among them. The inter-domain protocol for the goal of DDoS protection coordination is the main focus of this document. Note that both network operators and cloud based DDoS protection service providers can participate in the inter-domain DDoS protection service. In general, the member operator's own DDoS protection system should at least consist of an attack detector, a customer (DOTS client), a controller (DOTS server, and possible DOTS client for the inter-domain use cases) and a mitigator.

Attack Detector: responsible for attack detection and source traceback. An example is the flow analyzer

Customer: when a DDoS attack is detected, it requests mitigation service to the controller and exchanges status with the controller regularly

Controller: responsible for intra-domain DDoS mitigation controlling and communication with the customer and other operators' controllers for inter-domain coordination

Mitigator: responsible for mitigation and results reporting

here are two ways for operators to implement the inter-domain DDoS protection service: distributed or centralized. The following sections discuss these architectures, aligning with DOTS terms.

4.1. Distributed Architecture

Operators can set up bilateral cooperative relationships of DDoS protection between each other, thereby a distributed inter-domain DDoS protection service is realized, which has peer to peer

communication among all the participating operators. The distributed architecture is illustrated in the following diagram:

```
                                                      Customer
                                                     +-------+
                                                     |DOTS   |
                                                     |Client |
                                                     +----A--+
                                                          |
                ----------------                        --+------
           ////-                \\\\                 ///  |      \\\
        ///                 +-----------------------------+-------+ \\
      //  +-----------------+-------+  \\         /       |       |   \
    ||    |                 |       |    ||         +-----V-+-----V-+  |
   | +----V----------+  +---V---+---V---+  |    |   |DOTS   |DOTS   |  |
   | |DOTS   |DOTS   <-->DOTS   |DOTS   <--+----+--->Server |Client |  |
   | |Server |Client |  |Server |Client |  |    |   +-----A-+------A+  |
   | +--A----+-------+  +----A--+---A---+  |     |    Controller  /    |
   |    | Controller       Controller\     |     |      /        /     |
    ||  |                    | \      \  ||       \   //       //     /
      \\|                    |  \      //          \\/   ISP2 /     //
        |\\        ISP1      |   \   //  \          / \\     /   ///
        |  \\\\-             | -////      \        /    ----/----
        |       -+-----------+-    \       \      /        /
        |                    |      \       \    /        /
        |                    |       \       \  /        /
    +---V---+           +----V--+     \   -----/---    //
    |DOTS   |           |DOTS   |      \//   / \\  \\ /
    |Client |           |Client |    // \   /    \   /\\
    +-------+           +-------+   /    \ /      \ /   \
    Customer            Customer      +---V---+----V--+  |     +-------+
                                  |   |DOTS   |DOTS   |  |     |DOTS   |
                                  |   |Client |Server <---+---->Client |
                                  |   +-------+-------+   |    +-------+
                                   |    Controller       |     Customer
                                   |                     |
                                    \   cloud based     /
                                     \\  Anti-DDoS    //
                                       \\\ Provider ///
                                          --------
```

Figure 1: Distributed Architecture for Inter-domain DDoS Protection Service

As shown in the above diagram, when a customer is suffering a large traffic DDoS attack, it acts as the DOTS client to request DDoS protection service from its operator. The operator's controller acts as the DOTS server to authenticate the customer's validity and then

initiate the intra-domain DDoS mitigation service with its own resource for the customer. If the controller finds the attack volume exceeds its capacity, or the attack type is unknown type, or its inter-domain upstream link is congested, it should act as the DOTS client to request inter-domain coordinated DDoS Protection service to its upstream operators' controllers which it has cooperative relationship with. The operator's controller should support the functions of DOTS server and DOTS client at the same time in order to participate in the system of inter-domain DDoS protection service. In other words, as the representative for an operator's DDoS protection service, the controller manages and provides DDoS mitigation service to its customer on one hand, but on the other, it may require help from other operators under other situations, especially when the attack volume exceeds its capacity or the attack is from other operators. The inter-domain coordination can be a repeated process until the operator nearest the attack source receives the inter-domain coordination request and starts to mitigate the attack traffic.

In particular, each operator is able to decide its own responding actions to its peering operator's request flexibly by its internal policies, such as whether or not perform the mitigation function, or relay the request message to other operators. These other scenarios are out of the scope of this document.

The distributed architecture is straightforward and simple when the number of member operators are not too large. For deployment, all the work an operator needs to do is to configure other cooperative member operator's information (i.e., IP, port, DNS name, etc) and relevant policies for subsequent inter-domain communication. Regarding operation, each operator's controller only performs the mitigation service according to customer's request and possibly requests for inter-domain help to other operators if necessary. In the meantime, the mitigation report and statistics information is exchanged between the peering operators for the purpose of monitoring and accounting.

Some points for this architecture are noted below:

- o Every operator controller only has the information of those operators which have cooperative relation with it, and does not necessarily have the information of all operators participating in the inter-domain DDoS protection service. The incomplete information may not lead to the most optimized operation.

- o When the number of member operators is very large, a new joining operator will be required to configure and maintain a large number

of peering operators' information. This can be very complex and error-prone.

- Due to the exclusive repeating nature of the architecture mentioned above, it's possible that a really effective mitigation service by one upstream operator starts only after several rounds of repeating the inter-domain coordination process. This process may take a long time and is unacceptable.

4.2. Centralized Architecture

For the centralized architecture, the biggest difference from the distributed architecture is that a centralized orchestrator exists for controlling the inter-domain DDoS protection coordination centrally. This centralized architecture for the inter-domain DDoS protection service is illustrated in the following diagram:

```
                                    Orchestrator
                                +-------+-------+
                                ADOTS   |DOTS   A
                               /|Server |Client |\
                             /  +---AA--+A--A---+  \
                           /        |  \  /          \
                         /          |/  /\            \
                       /           /| /    \           \
                -----/---------- /  /       \           --\------
           ////-   /           /\\\\|         \      ///   \     \\\
        ///      /           /   /  |\\        \   //        \      \\
      //       /           /   /    |  \\       \/            \      \
    ||       /           /   /      |    ||       \ +-------+---V---+  |
   | +-----V---------+ /+---V---+---V---+  |    |  \|DOTS   |DOTS   |  |
   | |DOTS   |DOTS   V  |DOTS   |DOTS   |  |    |   VClient |Server |  |
   | |Client |Server |  |Server |Client |   |   |   +-------+---A---+  |
   | +-------+---A---+  +----A--+-------+  |     |   Controller |      |
   |  Controller |           |  Controller |     |              |      |
    ||           |           |           ||       \             |     /
      \\         |           |         //          \\    ISP2   |   //
        \\\      | ISP1      |       //              \\\        |///
           \\\\- |           | -////                    --------+
                -+-----------+-                                 |
                 |           |                                  |
                 |           |                                  |
           +-----V-+    +----V--+                           +---V---+
           |DOTS   |    |DOTS   |                           |DOTS   |
           |Client |    |Client |                           |Client |
           +-------+    +-------+                           +-------+
           Customer     Customer                            Customer
```

Figure 2: Centralized Architecture for Inter-domain DDoS Protection Service

As shown in above diagram, the orchestrator is the core component to the whole system. Each operator controller only communicates with it for the goal of registering, coordination requesting and reporting. When it receives the inter-domain coordination request message from the operator controller, a simple way is to notify all the other operator controllers that have registered to the orchestrator, to enable possible mitigation services. Another way is to choose a number of operators to notify them to enable the mitigation services according to the traceback result or other policies. The details about traceback are to be discussed in future. Based on the above analysis, the orchestrator is also a combination of a DOTS server and a DOTS client which supports both functions at the same time.

In addition to the orchestrator and its related functions, the signaling and operations of the centralized architecture are very similar to the distributed architecture.

The centralized architecture has its own unique characteristics described below:

o Since this is a centralized architecture, it is easy for the orchestrator to suffer a single failure problem like failure, congestion or performance downgrade, which would directly influence the availability of the whole system. This issue can be improved somewhat by implementing some redundancy mechanisms.

o A centralized orchestrator facilitates the auto-discovery mechanism for the member operators. And for each controller, its deployment and operation becomes easy since it is only required to communicate with the orchestrator during the whole process.

o Due to the direct communication between the orchestrator and all controllers, the inter-domain DDoS coordination is able to be finished in a short and fixed time period.

o Only the central orchestrator is required to support different transport protocols (e.g., TCP, UDP, CoAP) to communicate with all the controllers. The orchestrator is able to translate and relay different transport protocols among all the operators. So, the operator controller uses one transport protocol to communicate with orchestrator and is not required to support multiple kinds of transport protocols.

5. Inter-domain DOTS Protocol

According to [I-D.draft-ietf-dots-requirements], DOTS protocols MUST take steps to protect the confidentiality, integrity and authenticity of messages sent between the DOTS client and server, and provide a peer mutual authentication between the DOTS client and server before a DOTS session is considered active. The DOTS agents can use HTTPS (with TLS) for the goal of protocol security. The HTTP RESTful APIs are used in this section as the protocol channel, and the DOTS message content can be in JSON format.

With respect to the inter-domain DOTS protocol, all the DOTS messages are exchanged between DOTS client and server, no matter what the architecture (distributed or centralized) is. Therefore, the message formats and operations of the DOTS protocol ought to be the same for all architecture options. The DOTS messages can be categorized by which time period they are mainly required for DDoS protection, as below:

- o Provisioning stage: Before being attacked by malicious traffic, a DOTS client should register itself to the DOTS server, as well as enable capacity building in advance.

- o Signaling stage: Once the DOTS client has registered itself to the DOTS server, the DOTS session is created between client and server and the signaling stage begins. The signaling stage ends when the DOTS client cancels its registration to the DOTS server and the DOTS session is closed. During the signaling stage, the DOTS client should ask the DOTS server for DDoS mitigation service to the customer service under attack once an attack is detected. When an attack is over, the DOTS server should notify the DOTS client.

DOTS protocol can run on HTTPS (with TLS) and support several different ways for authentication:

- o Employ bidirectional certificate authentication ([ITU-T X.509]) on the DOTS server and the client: Both DOTS server and client MUST verify the certificates of each other.

- o Employ unidirectional certificate authentication ([ITU-T X.509]) on the DOTS server: Only the DOTS server needs to install the certificate. The DOTS client only needs to verify its certificate. In the opposite direction, the DOTS server can authenticate the DOTS client by the ways of a user/role:password, IP address white-list or digital signature.

- o Employ bidirectional digital signature authentication on the DOTS server and client: In this scenario, the DOTS server and client must keep the customer's private key safe. This private key is used to generate the digital signature.

Besides authenticating the DOTS client, the DOTS server also verifies the timestamp of the packets from the DOTS client. If the time difference between the timestamp and the current time of the DOTS server exceeds a specified threshold (60 seconds as an example), the DOTS server will consider the packet invalid and will not process it. Therefore, NTP must be configured on both the DOTS server and client to ensure time synchronization. This method can effectively protect the DOTS server against a replay attack.

The following sections present detailed description of all the DOTS messages for each stage, and the relevant DOTS protocol operations.

5.1. Provisioning Stage

In the provisioning stage, a DOTS client can be located either in the customer side, in the operator controller, or in the inter-domain orchestrator (for the centralized architecture). In any case, the DOTS client should register itself to its peering DOTS server which provides the intra/inter domain DDoS mitigation service to it in order to set up the DOTS protocol session. More importantly, the registration process also facilitates the auto-discovery, capacity building and configuration between the DOTS client and server.

5.1.1. Messages

In the provisioning stage, the messages of registration (DOTS client to server), registration response (DOTS server to client), registration cancelling (DOTS client to server) and registration cancelling response (DOTS server to client) are required. Since all the messages in this stage are not expected to be used under the DDoS attack conditions, transmitting all the messages through DOTS data channel over TLS is able to meet the requirements of reliability, privacy and integrity.

The HTTP POST method with the message body in JSON format is used for the registration and registration response messages as below:

```
METHOD:POST - URL:{scheme}://{host}:{port}/dots/api/registration
registration body:
{
 "customer_name": string;
 "ip_version": string;
 "protected_zone": {
      "index": number;
      "need_alias": string;
      "ipv4_CIDR": string;
      "ipv6_address": string;
      "BGP_route": string;
      "SIP_URI": string;
      "E164_number": string;
      "DNS_name": string;
 }
 "protected_port": string;
 "protected_protocol": string;
 "countermeasures": string;
 "tunnel_information": string;
 "next_hop": string;
 "security_profile": {
     "TLS": string;
     "DTLS": string;
```

```
            "CoAP": string;
        }
        "white_list": {
            "name": string;
            "sequence_number": string;
            "source_ip": string;
            "destination_ip": string;
            "source_port": string;
            "destination_port": string;
            "protocol": string;
            "length": string;
            "TTL": string;
            "DSCP": number;
            "ip_flags": number;
            "tcp_flags": number;
        }
        "black_list": {
            "name": string;
            "sequence_number": string;
            "source_ip": string;
            "destination_ip": string;
            "source_port": string;
            "destination_port": string;
            "protocol": string;
            "length": string;
            "TTL": string;
            "DSCP": number;
            "ip_flags": number;
            "tcp_flags": number;
        }
       }
       registration response body:
       {
        "customer_name": string;
        "customer_id": string;
        "alias_of_mitigation_address": {
            "index": number;
            "alias": string;
        }
        "security_profile": string;
        "access_token": string;
        "thresholds_bps": number;
        "thresholds_pps": number;
        "duration": number;
        "capable_attack_type": string;
        "registration_time": string;
        "mitigation_status": string;
        }
```

Registration body:
customer_name: The name of the customer (DOTS client);
ip_version: Current IP version. It can be "v4" or "v6";
protected_zone: Limit the address range of protection. Especially it will be limited to the prefixes possessed by the customer;
index: index of the protected zone;
need_alias: the flag representing if this protected zone needs an alias. "true" represents that the alias is needed, "false" represents the opposite side;
ipv4_CIDR: ipv4 CIDR address or prefix scope of the protected zone;
ipv6_address: ipv6 address or prefix scope of the protected zone;
BGP_route: BGP route of the protected zone;
SIP_URI: SIP URI of the protected zone;
E164_number: E.164 number of the protected zone;
DNS_name: DNS name of the protected zone;
protected_port: Limit the port range of protection, "all" represents all the ports are to be protected;
protected_protocol: The protected protocol indicated by the protocol attribute in the IP packet header, "all" represents all the protocols are to be protected;
countermeasures: Some of the protection need mitigation and others need blackholing;
tunnel_information: The tunnel between the mitigation provider's network and the customer's network. Tunnel technologies such as GRE[RFC2784] can be used to
return normal traffic. "null" represents there is no tunnel information provided and the DOTS server can decide the return tunnel for the
normal traffic for itself;
next_hop: The returning path to customer's network. "null" represents there is no next hop information provided and the DOTS server can decide it for itself;
security_profile: The security profile in transport layer for the DOTS signaling channel that DOTS client supports;
TLS: "true" represents that the DOTS client supports TLS over TCP, "false" represents the opposite side;
DTLS: "true" represents that the DOTS client supports DTLS over UDP, "false" represents the opposite side;
CoAP: "true" represents that the DOTS client supports CoAP, "false" represents the opposite side;
white_list: The white-list information provided to the DOTS server;
name: Name of the white-list;
sequence_number: Sequence number of the white-list;
source_ip: The source IP address attribute used in the white-list;
destination_ip: The destination IP address attribute used in the white-list;
source_port: The source port attribute used in the white-list;
destination_port": The destination port attribute used in the white-list;

protocol: The protocol attribute in the IP packet header used in the white-list;
length: The length attribute in the IP packet header used in the white-list;
TTL: The TTL attribute in the IP packet header used in the white-list;
DSCP: The DSCP attribute in the IP packet header used in the white-list;
ip_flags: The IP flags attribute used in the white-list;
tcp_flags: The TCP flags attribute used in the white-list;

black_list: The black-list information provided to the DOTS server;
name: Name of the black-list;
sequence_number: Sequence number of the black-list;
source_ip: The source IP address attribute used in the black-list;
destination_ip: The destination IP address attribute used in the black-list;
source_port: The source port attribute used in the black-list;
destination_port: The destination port attribute used in the black-list;
protocol: The protocol attribute in the IP packet header used in the black-list;
length: The length attribute in the IP packet header used in the black-list;
TTL: The TTL attribute in the IP packet header used in the black-list;
DSCP: The DSCP attribute in the IP packet header used in the black-list;

ip_flags: The IP flags attribute used in the black-list;
tcp_flags: The TCP flags attribute used in the black-list;

registration response body:
customer_name: The name of the customer (DOTS client);
customer_id: The unique id of the customer (DOTS client);
alias_of_mitigation_address:
    index: index of the protected zone;
    alias: The alias that the DOTS server assigns to this protected zone;
security_profile: The negotiated security profile for the DOTS session;
access_token: Authentication token (e.g. pre-shared nonce). "null" represents there is no access token;
thresholds_bps: If an attack volume is over this threshold, the controller will reject the protection in order to comply with the negotiated contract;
thresholds_pps: If an attack volume is over this threshold, the controller will reject the protection in order to comply with the negotiated contract;
duration: If an attack is longer than this threshold, the controller will reject the protection in order to comply with the negotiated contract;
capable_attack_type: Limit the protectable attack type;
registration_time: The time of registration;
mitigation_status: The status of current mitigation service of the ISP.

Similarly, another HTTP POST method with the message body in JSON format is used for the registration cancelling and registration cancelling response messages are described below:

```
METHOD:POST - URL:{scheme}://{host}:{port}/dots/api/registration_cancelling
   registration cancelling body:
   {
   "customer_id": string;
   "reasons": string;
   }
   registration cancelling response body:
   {
   "customer_id": string;
   "result": string;
   }
```

Registration cancelling body:
customer_id: The unique id of the customer (DOTS client);
reasons: The reasons why the DOTS client cancelled the registration;

registration cancelling response body:
customer_id: The unique id of the customer (DOTS client);

result: The final result which defines whether or not the DOTS controller accepts the registration cancelling request.

5.1.2. Operations

The main operations in the provisioning stage include:

o The customers (DOTS client) registers to the operator controller with the configuration and capability building including protection methods, process capacity, protected zone, security profile, white/black-list, etc;

o The DOTS client in operator controller registers to the DOTS server in inter-domain orchestrator (centralized architecture) or other operator controllers (distributed architecture) according to inter-domain DDoS protection requirements;

o The DOTS client can send the registration cancelling message to the DOTS server for cancelling its DDoS protection service.

The DOTS server indicates the result of processing the POST request using HTTP response codes:

o Response code 200 (OK) will be returned in the response if the DOTS server has accepted the mitigation request and will try to mitigate the attack. The HTTP response will include the JSON body of response messages specified above;

o If the request is missing one or more mandatory attributes then 400 (Bad Request) will be returned in the response or if the request contains invalid or unknown parameters then 500 (Invalid query) will be returned in the response. The HTTP response will include the JSON body received in the request, with an extra attribute to represent the specific error reason:

```
     "error_reason": number;
        0: Bad Request;
        1: Invalid Query;
        2: Server Error;
        3: Protected Zone Confliction;
        4: Countermeasure Not Supported;
        5: Security Profile Not Supported;
        6: Confliction Exists for White-list or Black-list;
        255: Others;
```

5.2. Signaling Stage

During the signaling stage, the DOTS signaling channel created with the negotiated security profile in the provisioning stage is used for the DDoS attack mitigation coordination. Once the DOTS client detects the attack to the customer service, a mitigation initiation request message is created and sent to the provisioned DOTS server to call for the DDoS protection service. The DOTS server decides to protect the customer service based on the information from the request message and its configured policy. One operator's DOTS

server may ask the co-located DOTS client to resume sending the mitigation initiation request message to other operators' DOTS server to request the inter-domain coordinated mitigation service while it isn't able to deal with the attack by itself. Meanwhile, some other messages are required to be communicated between the DOTS client and server for information updates about status, efficacy and scope. When the DOTS server is informed from the mitigator that the attack is over, it should notify the DOTS client to terminate the mitigation service.

5.2.1. Messages

In the signaling stage, the DOTS signaling channel is expected to transmit DOTS messages under extremely hostile network conditions such as link saturation. To meet the requirements of resilience and robustness, unidirectional messages MUST be supported within the bidirectional signal channel to allow for unsolicited message delivery, enabling asynchronous notifications between the DOTS client and server. So, the listed DOTS messages are required: mitigation initiation request (DOTS client to server), mitigation efficacy updates (DOTS client to server), mitigation status updates (DOTS server to client), mitigation termination (DOTS client to server), mitigation termination status acknowledgement (DOTS client to server) and heartbeat (bidirectional message).

Mitigation Request:

A HTTP POST method with the message body in JSON is used for the mitigation request message:

```
METHOD:POST - URL:{scheme}://{host}:{port}/dots/api/mitigation_request
  mitigation request body:
  {
   "version": string;
   "type": string;
   "alert_id": string;
   "sender_id": string;
   "sender_asn": string;
   "mitigation_action": number;
   "lifetime": number;
   "max_bandwidth": number;
   "packet_header": {
       "dst_ip": string;
       "alias": string;
       "dst_ports": string;
       "src_ips": string;
       "src_ports": string;
       "protocols": string;
```

```
              "tcp_flags": string;
              "fragment": string;
              "pkt_len": string;
              "icmp_type": string;
              "icmp_code": string;
              "DSCP": string;
              "TTL": string;
          }
          "current_throughputs": {
              "bps": string;
              "pps": string;
          }
          "peak_throughputs": {
              "bps": string;
              "pps": string;
          }
          "average_throughputs": {
              "bps": string;
              "pps": string;
          }
          "info": {
              "attack_types": string;
              "started": number;
              "ongoing": number;
              "severity": number;
              "direction": number;
              "health": number;
          }
          "vendor": {
              "name": string;
              "version": string;
              "payload": {
                     "offset": number;
                     "content": string;
                     "hash": string;
              }
           }
         }
```


mitigation request body:
version: A 3 digit set, similar to Linux. (Major.Minor.Revision);
type: Only "attack" in scope for v1;
alert_id: A SHA-256 hash that is derived from DST_IP and started with some random nonce;
sender_id: A SHA-256 hash signature of the sender. This is used to validate who sent it;

sender_asn: ASN of the sender. This could be used to link back to sender_id to validate the sender of being a valid sender_id;

mitigation_action: The requested mitigation actions by DOTS client.

Possible value could be: 1 - mitigation, 2 - blackhole, 3 - flowspec, ...;
lifetime: The desired lifetime of the mitigation service from the DOTS client. Upon the expiry of this lifetime, and if the request is not refreshed, the
mitigation service is stopped. The service can be refreshed by sending the message with the same "alert_id" again. A lifetime of zero indicates
indefinite lifetime for the mitigation service. This is an optional attribute in the request message;
max_bandwidth: The max bandwidth the DOTS client can undertake. The unit is "G bytes";
packet_header: IP packet header contents used for a report. CSV (Comma Separated Values) format is used here when multiple values are possible. Note that no
spaces between commas for CSV format, and the multiple values for every attribute should be in the same order as they are assigned.
dst_ip: A single IP under attack;
alias: The DOTS client's registered alias for the protected zone;
dst_ports: The destination port(s) used for the attack. CSV formatted;
src_ips: The list of source IPs of the attack. CSV formatted;
src_ports: The source port(s) used for the attack. CSV formatted;
protocols: The IP protocol numbers used for the attack. The list of IP protocol numbers are defined and maintained by IANA. CSV formatted;
tcp_flags: The TCP flags used for the attack. Possible value could be: SYN, FIN, ACK, PSH, RST, URG, NULL. CSV formatted;
fragment: The fragment flags in the IP header for the attack. Possible value could be: DF - Don't fragment, IsF - Is a fragment, FF - First fragment,
LF - Last fragment. CSV formatted;
pkt_len: The packet length used for the attack. CSV formatted;
icmp_type: The icmp type used for the attack. CSV formatted;
icmp_code: The icmp code used for the attack. CSV formatted;
DSCP: The DSCP value used for the attack. CSV formatted;
TTL: The TTL value used for the attack. CSV formatted;
current_throughputs: Current throughput in bps/pps for the above attack flows
bps: bytes per second. CSV formatted;
pps: packets per second. CSV formatted;
peak_throughputs: The peak throughput in bps/pps for the above attack flows until the time the DOTS request message is sent
bps: bytes per second. CSV formatted;
pps: packets per second. CSV formatted;
average_throughputs: The calculated average throughput in bps/pps for the above attack flows until the time the DOTS request message is sent
bps: bytes per second. CSV formatted;
pps: packets per second. CSV formatted;

```
         info: Other general information which may be useful
            attack_types: List of attacks being used together for this attack,
on this single DST_IP. CSV formatted;
            started:  Unix EPOCH when the attack is started;
            ongoing: The value representing whether the attack is still
ongoning. 1 - yes, 0 - no;
            severity: The severity level of the attack. 1, 2, 3 - low, medium,
high;
            direction: The direction of the attack. in or out;
            health: The health condition of the DOTS client. 0-100;
         vendor:
            name: Company name;
            version: version of the DOTS client on the vendors device;
            payload: The attack packet payload provided to DOTS server for
further analysis
               offset: The payload offset;
               content: The payload content that is base64 encoded;
            hash:  A SHA-256 hash used as a checksum, of the original payload
before being base64 encoded. This is to proof the payload is complete. Not to
prove if
                   it has been tampered with;
```

Mitigation Status Exchange:

A HTTP POST method with the message body in JSON is used for the mitigation efficacy updates message:

```
       METHOD:POST - URL:{scheme}://{host}:{port}/dots/api/
mitigation_efficacy_updates
         mitigation efficacy updates body:
         {
          "version": string;
          "alert_id": string;
          "sender_id": string;
          "sender_asn": string;
          "attack_status": string;
          "health": number;
         }
```

mitigation efficacy updates body:
version: A 3 digit set, similar to linux. (Major.Minor.Revision);
alert_id: A SHA-256 hash that is derived from DST_IP and started with some random nonce;
sender_id: A SHA-256 hash signature of the sender. This is used to validate who sent it;
sender_asn: ASN of the sender. Could be used to link back to sender_id to validate the sender of being a valid sender_id;
attack_status: The current attack status of the DOTS client. Possible value could be: 0 - in-process, 1 - terminated;
health: The health condition of the DOTS client. 0-100;

A HTTP POST method with the message body in JSON is used for the mitigation status updates message:

```
      METHOD:POST - URL:{scheme}://{host}:{port}/dots/api/
mitigation_status_updates
      mitigation status updates body:
     {
      "version": string;
      "alert_id": string;
      "sender_id": string;
      "sender_asn": string;
      "status": number;
      "error_reason": number;
      "lifetime": number;
      "source_ports": string;
      "destination_ports": string;
      "source_ips": string;
      "destination_ip": string;
      "TCP_flags": string;
      "start_time": number;
      "end_time": number;
```

```
"forwarded_total_packets": number;
"forwarded_total_bits": number;
"forwarded_peak_pps": number;
"forwarded_peak_bps": number;
"forwarded_average_pps": number;
"forwarded_average_bps": number;
```

```
      "malicious_total_packets": number;
      "malicious_total_bits": number;
      "malicious_peak_pps": number;
      "malicious_peak_bps": number;
      "malicious_average_pps": number;
      "malicious_average_bps": number;
      "record_time": string;
      }
```

mitigation status updates body:

version: A 3 digit set, similar to Linux. (Major.Minor.Revision);

alert_id: A SHA-256 hash that is derived from DST_IP and started with some random nonce;

sender_id: A SHA-256 hash signature of the sender. This is used to validate who sent it. The sender is the DOTS server for this message;

sender_asn: ASN of the sender. This could be used to link back to sender_id to validate the sender of being a valid sender_id;

status: Current mitigation status, such as: pending, ongoing, done, error;

error_reason: If status attribute is error, then this attribute expresses its reason, the possible value could be: 0 - Bad Request, 1 - Server Error, 3 - Mitigation Scope Confliction, 4 - Mitigation Action Not Support, 255 - Others;

lifetime: The lifetime of mitigation service that DOTS server has assigned to DOTS client. DOTS client MUST follow this value;

source_ports: For TCP or UDP or SCTP or DCCP: the source range of ports (e.g., 1024-65535) of the discarded traffic. CSV formatted;

destination_ports: For TCP or UDP or SCTP or DCCP: the destination range of ports (e.g., 1-443) of the discarded traffic. CSV formatted;

source_ips: The source IP addresses or prefixes of the discarded traffic. CSV formatted;

destination_ip: The destination IP addresses or prefixes of the discarded traffic;

TCP_flags: TCP flag of the discarded traffic. CSV formatted;

start_time: The start time for the duration of this mitigation status message;

end_time: The end time for the duration of this mitigation status message;

forwarded_total_packets: The total number of packets forwarded;

forwarded_total_bits: The total bits for all the packets forwarded;

forwarded_peak_pps: The peak pps of the traffic forwarded;

forwarded_peak_bps: The peak bps of the traffic forwarded;

forwarded_average_pps: The average pps of the traffic forwarded;

forwarded_average_bps: The average bps of the traffic forwarded;

malicious_total_packets: The total number of malicious packets;

malicious_total_bits: The total bits of malicious packets;

malicious_peak_pps: The peak pps of the malicious traffic;

malicious_peak_bps: The peak bps of the malicious traffic;
malicious_average_pps: The average pps of the malicious traffic;
malicious_average_bps: The average bps of the malicious traffic;
record_time: The time that the mitigation status updates message is created;

Mitigation Termination:

A HTTP POST method with the message body in JSON is used for the mitigation termination request message:

```
      METHOD:POST - URL:{scheme}://{host}:{port}/dots/api/
mitigation_termination_request
         mitigation termination request body:
         {
          "version": string;
          "alert_id": string;
          "sender_id": string;
          "sender_asn": string;
         }
```

mitigation termination request body:
version: A 3 digit set, similar to linux. (Major.Minor.Revision);
alert_id: A SHA-256 hash that is derived from DST_IP and started with some random nonce;
sender_id: A SHA-256 hash signature of the sender. This is used to validate who sent it;
sender_asn: ASN of the sender. This could be used to link back to sender_id to validate the sender of being a valid sender_id;

A HTTP POST method with the message body in JSON is used for the mitigation termination status acknowledgement message:

```
        METHOD:POST - URL:{scheme}://{host}:{port}/dots/api/
mitigation_termination_status_acknowledgement
         mitigation termination status acknowledgement body:
         {
          "version": string;
          "alert_id": string;
          "sender_id": string;
          "sender_asn": string;
         }
```

mitigation termination status acknowledgement body:
version: A 3 digit set, similar to Linux. (Major.Minor.Revision);
alert_id: A SHA-256 hash that is derived from DST_IP and started with some random nonce;
sender_id: A SHA-256 hash signature of the sender. This is used to validate who sent it;
sender_asn: ASN of the sender. This could be used to link back to sender_id to validate the sender of being a valid sender_id;

Heartbeat:

A HTTP POST method with the message body in JSON is used for the heartbeat message:

```
       METHOD:POST - URL:{scheme}://{host}:{port}/dots/api/heartbeat
         heartbeat body
         {
          "version": string;
          "sender_id": string;
          "sender_asn": string;
         }
```

heartbeat body:
version: A 3 digit set, similar to Linux. (Major.Minor.Revision);
sender_id: A SHA-256 hash signature of the sender. This is used to validate who sent it;
sender_asn: ASN of the sender. This could be used to link back to sender_id to validate the sender of being a valid sender_id;

5.2.2. Operations

The main operations in the signaling stage include:

- The customer (DOTS client) detects a malicious attack, requests mitigation service to its operator controller (DOTS server);

- DOTS server authenticates and provides its intra- domain mitigation service to the DOTS client;

- When the DOTS server is mitigating the attack but finds that the attack volume exceeds its capacity, or the attack type is an unknown type, or its upstream link is congested, it should request to other DOTS servers for inter-domain cooperation;

- Working DOTS server report their statistics results by mitigation status updates message to the DOTS client;

- The DOTS client can updates its mitigation scope to the DOTS server by resending the mitigation request message. It also can update its mitigation efficacy result to the DOTS server;

- When the DOTS server is informed from the mitigator that the attack is over, it should notify the DOTS client by the mitigation status updates message to terminate the mitigation service;

- When the DOTS client is notified by the DOTS server to terminate its mitigation service, it should send a DOTS termination request message to the DOTS server. The DOTS server stops its mitigation service and notifies it to the DOTS client by sending DOTS status updates message. At last, the DOTS client sends a DOTS mitigation termination acknowledgement message to finish the whole DOTS session;

o The heartbeat message is exchanged between the DOTS client and the DOTS server to check their respective status. If any side of the channel fails to receive the heartbeat message, then it will trigger an alert or further investigation into why the heartbeats never reached their destination.

6. Other Considerations

6.1. Billing Data

This is not a technical issue nor a part of the DOTS protocol but it is relevant to deployment models. If other organizations utilized the resources of a DDoS protection service, it is natural to charge it according to the amount of use. However, how does one count the amount of use among different DDoS protection service providers. For example, some DDoS protection service provider charges users by volume of the attack traffic or dropped packets. On the other hand, some of them use the volume of normal traffic. The number of executions can be also used. We cannot decide what information should be taken into account for billing purposes in advance, however information is needed to be exchanged while coordinating DDoS protection. This information could be also used to determine which service would be used when asking for help. Though it is out of the scope for DOTS, coordinating and optimizing this cooperation this business aspect is difficult to solve.

7. Security Considerations

TBD

8. IANA Considerations

No need to describe any request regarding number assignment.

9. Normative References

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

[RFC2784] D. Farinacci., T. Li., S. Hanks., D. Meyer., and P. Traina., "Generic Routing Encapsulation (GRE), March 2000".

[I-D.draft-ietf-dots-use-cases]
R. Dobbins, Ed., S. Fouant., D. Migault., R. Moskowitz., N. Teague., L. Xia, "Use cases for DDoS Open Threat Signaling, October 2015".

[I-D.draft-ietf-dots-requirements]
A. Mortensen., R. Moskowitz., and T. Reddy., "DDoS Open Threat Signaling Requirements, draft-ietf-dots-requirements-00, October 2015".

[I-D.draft-mortensen-dots-architecture]
A. Mortensen., F. Andreasen., T. Reddy., C. Gray., R. Compton., and N. Teague., "Distributed-Denial-of-Service (DDoS) Open Threat Signaling Architecture, March 2016".

[I-D.draft-reddy-dots-transport]
T. Reddy., D. Wing., P. Patil., M. Geller., M. Boucadair., and R. Moskowitz., "Co-operative DDoS Mitigation, June 2016".

Authors' Addresses

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

EMail: kaname@nttv6.jp

Liang Xia
Huawei Technologies Co., Ltd.
101 Software Avenue, Yuhuatai District
Nanjing, Jiangsu
210012, China

EMail: frank.xialiang@huawei.com

Jinwei Xia
Huawei Technologies Co., Ltd.
101 Software Avenue, Yuhuatai District
Nanjing, Jiangsu
210012, China

EMail: xiajinwei@huawei.com

Dacheng Zhang
Beijing
China

EMail: dacheng.zdc@alibaba-inc.com

Luyuan Fang
Microsoft
15590 NE 31st St
Redmond, WA 98052

EMail: lufang@microsoft.com

Christopher Gray
Comcast, Inc.
United States

EMail: Christopher_Gray3@cable.comcast.com

Rich Compton
Charter Communications, Inc.
14810 Grasslands Dr
Englewood, CO 80112

EMail: Rich.Compton@charter.com