DOTS
Internet-Draft
Intended status: Standards Track
Expires: June 30, 2017

K. Nishizuka
NTT Communications
L. Xia
J. Xia
Huawei Technologies Co., Ltd.
D. Zhang

L. Fang
Microsoft
C. Gray
Comcast, Inc.
R. Compton
Charter Communications, Inc.
December 27, 2016

Inter-organization cooperative DDoS protection mechanism draft-nishizuka-dots-inter-domain-mechanism-02

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on June 30, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

These days, DDoS attacks are getting bigger and more sophisticated. All organizations facing to the internet should be prepared for such attacks in order to minimize damages to their business. There are still too big platforms of DDoS attacks which consist of vulnerable servers, broadband routers and other network equipments including IoT devices 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 attack is commoditized, so frequency of DDoS attacks is also increasing.

These trends of the attack could exceed a capacity of a protection system of one organization in the aspect of volume and frequency. Therefore, sharing the capacity and capability of the protection system with each other has become very necessary.

By utilizing other organization's resources, the burden of the protection can be 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 protection which utilize resources of each other "cooperative DDoS protection".

The cooperative DDoS protection have numerous merits. First, as described above, it can leverage the capacity of the protection by sharing the resources among organizations. Generally DDoS attack happens 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 other organization which is under attack. Second, organizations can get various countermeasures. If an attack is highly sophisticated and there is no countermeasure in the system, cooperative DDoS protection can offer optimal countermeasure of all partners. Third, it can block malicious traffic near 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 are mostly consist of useless massive attack traffic. Sometimes uplink circuits get congested by massive attacks, so cooperative DDoS protection by uplink organization is the only way to solve such a congestion problem. Finally, it can reduce time to respond. After getting attacked, prompt response is important because the outage of the service can make significant loss to the victim organization. Cooperating channel between partner organizations can be automated by DOTS protocol.

The proposed solutions are covering both intra-organization and inter-organization situations.

1.1. Scope

The solutions described in this draft are based on intra-organization and inter-organization usecases in [I-D.draft-ietf-dots-use-cases]. The DOTS protocols coordinating DDoS protection in inter-organization 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 organization scenarios are also valuable[I-D.draft-mortensen-dots-architecture]. The data model described in this draft is mainly focusing on inter-organization coordination of DDoS protection because it also covers single organization scenarios. The information required in single organization scenarios is assumed to be a subset of the information required in inter-organization 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. Inter-organization DDoS Protection Requirements

In this section, requirements unique to inter-organization cooperative DDoS protection are described.

3.1. Provisioning Requirements

In inter-organization 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 easier way to utilize DDoS protection service of other organizations. An organization can trust other organization who are going to use their DDoS protection service by any means like phone, e-mail, Web portal, etc,. However, it will take much time to provision the DDoS protection system, therefore the attack will succeed to make significant impact on the protected service. To reduce the time to start the protection, automatic provisioning is desirable. If an organization could build a capacity of protection by exchanging information of the DDoS protection service via dots protocol in short time, the cooperative DDoS protection will succeed at a certain level. Other important works carried out in the bootstrapping process are auto-discovery, 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 provider of cooperative DDoS protection is important factor. If only two providers are involved, there is a bilateral relationship only. It is easy to negotiate about the capacity of their own DDoS protection system. In the state of emergency, they can decide to ask for help each other if the capacity of their 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 about their capacity with every participant. It is needed to take into account all combinations to do appropriate protection. The coordination between the member providers of cooperative DDoS protection is a complete process consisting of mitigation start/stop, status notification, mitigation policy updates and so on. The Inter-organization DOTS architectures described in the later section are intended to fulfill these requirements.

3.2.1. Near Source Protection Problem

Stopping malicious traffic at the nearest point in the internet will reduce exhaustion of resources in all the path of the attack. To find the entry point of the attack, traceback of the attack traffic to the origin is needed. If there is cooperative partner near the attack source, asking for help to that network is most effective. However, the problem is that it is difficult to decide which network is nearest to the attack source because in many cases source address of attack packets are spoofed to avoid to be visible from others.

Moreover, uncovering some topology information of operator's network would be required in order to make the decision correctly, however there could be privacy protection issue between operators. Those problems will lead to the difficulties of locating the attack source.

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 should be returned to the original destination. The returning path is often called "clean pipe". The DDoS service provider should be careful about routing loop because if the end point of a clean pipe is still included in a reach of the announced BGP route, the traffic will return to the mitigation path again and again. In the case of inter-organization DDoS protection, returning path information should be propagated to partners.

4. Inter-organization 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-organization uplink congestion problem, but also take full advantage of global DDoS mitigation resources from different operators efficiently and enable the near source mitigation. Moreover, with the way of providing DDoS mitigation as service, more customers will get the service flexibly by their demands with maximized territory and resources. Together with on- premise DDoS protection appliance, 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.

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, and describes their communications. Although this document is initiated to specify the DOTS protocol for the inter-organization 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-organization use cases. To support all the identified use cases and possibly new use cases in future, DOTS protocol must be extensible in terms of the message definition, protocol process, etc, which will be discussed in details in the following section. The text below discusses the protocol mainly in respect of inter-organization use cases.

The inter-organization DDoS protection service is set up by the member operators' own DDoS protection system and the coordination protocol between them. The inter-organization 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-organization DDoS protection service. In general, the member operator's own DDoS protection system should at least consist of attack detector, customer (DOTS client), controller (DOTS server, and possible DOTS client for the inter-organization use cases) and mitigator.

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

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

Controller: be responsible for intra-organization DDoS mitigation controlling and communication with customer and other operators' controller for inter-organization coordination

Mitigator: be responsible for mitigation and results reporting

There are two ways for operators to implement the inter-organization DDoS protection service: distributed way or centralized way. The following parts give the respective discussion to them with aligning to DOTS terms.

4.1. Distributed Architecture

Operators can set up the bilateral cooperative relation of DDoS protection between each other, thereby a distributed inter-organization DDoS protection service is realized, which has the peer to peer Communication among all the participated operators. The distributed architecture is illustrated in the following diagram:

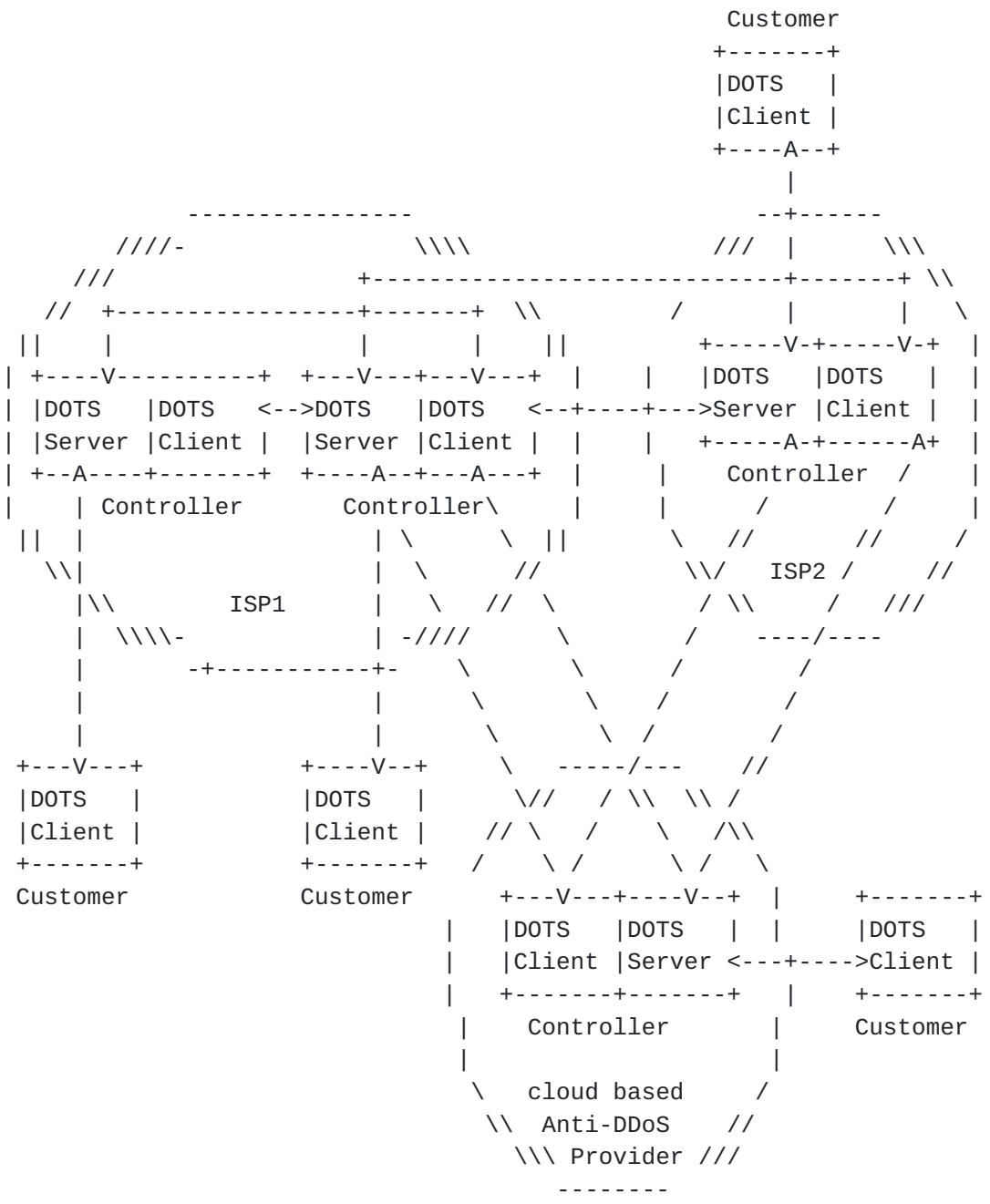

Figure 1: Distributed Architecture for Inter-organization DDoS Protection Service

As shown in above diagram, when the customer is suffering a large traffic DDoS attack, it acts as the DOTS client to request DDoS protection service to its operator. The operator's controller acts as the DOTS server to authenticate the customer's validity and then initiate the intra-organization DDoS mitigation service with its own resource for the customer. If the controller finds the attack volume exceeds it capacity, or the attack type is unknown type, or its

inter-organization upstream link is congested, it should act as the DOTS client to request inter-organization coordinated DDoS Protection service to its upstream operators' controller which it has cooperative relation 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-organization 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 in one hand, but may require helps from other operators under some situation especially when the attack volume exceeds its capacity or the attack is from other operators. The inter-organization coordination can be a repeated process until the nearest operator located from the attack source receives the inter-organization coordination request and starts to mitigate the attack traffic.

In particular, each operator is able to decide its 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. But these 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-organization communication. Regarding to operation, each operator's controller just performs the mitigation service according to customer's request and possibly requests for inter-organization helps to other operators if necessary. In the meantime, the mitigation report and statistics information is exchanged between the peering operators for the goal of monitoring and accounting.

Some points for this architecture are noted as below:

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

- When the number of member operators is very large, a new joining operator will be required to configure and maintain a lot of peering operators' information. It's very complex and error-prone

- Due to the exclusive repeating nature of the this architecture mentioned above, it's possible that the really effective

mitigation service by one upstream operator starts after several rounds of repeating the inter-organization coordination process. It 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-organization DDoS protection coordination centrally. The centralized architecture for the inter-organization DDoS protection service is illustrated in the following diagram:

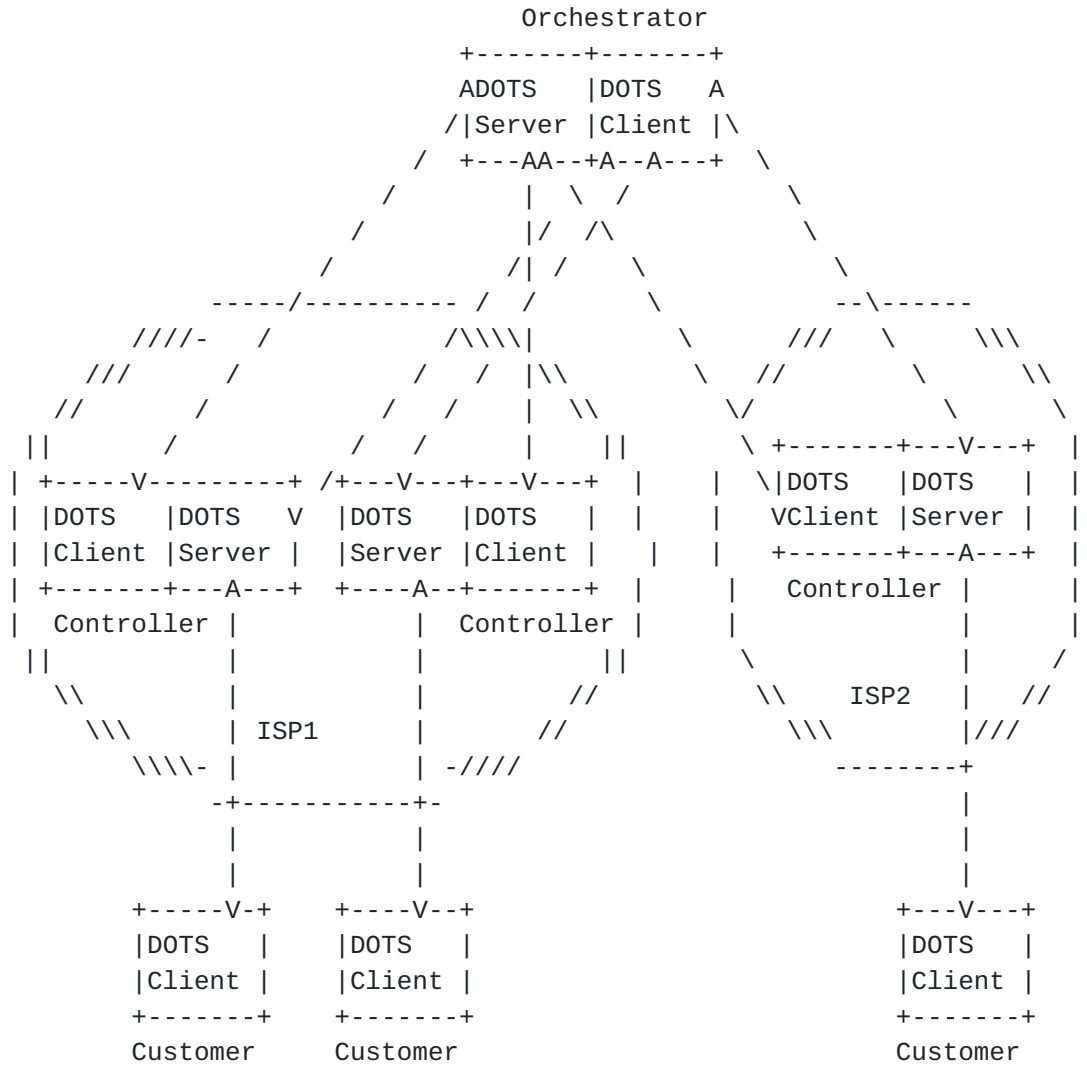

Figure 2: Centralized Architecture for Inter-organization DDoS Protection Service

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-organization coordination request message from the operator controller, a simple way is to notify all the other operator controllers which have registered to the orchestrator, to enable the 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 DOTS server and DOTS client which support both functions at the same time.

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

The centralized architecture has its own characteristics as below:

- Since it is the centralized architecture, the orchestrator is easy to suffer the single failure problem like failure, congestion and performance downgrade, which directly influences the availability of the whole system. This can be improved by some redundancy mechanism

- 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

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

- 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 protocol

5. Inter-organization 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 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-organization DOTS protocol, all the DOTS messages are exchanged between DOTS client and server, no matter what the architecture (distributed or centralized) is. So, the message formats and operations of 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 in for DDoS protection, as below:

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

- Signaling stage: Once the DOTS client has registered itself to the DOTS server, the DOTS session is created between them 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 the attack is detected. When the attack is over, the DOTS client should notify the DOTS server to stop the mitigation service.

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

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

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

- Employ bidirectional digital signature authentication on the DOTS server and client: In this condition, the DOTS server and client must keep the customer's private key safely, which is used for calculate 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 the 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 protect DOTS server against the replay attack effectively.

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

5.1. Provisioning Stage

In the provisioning stage, DOTS client can be located either in the customer side, in the operator controller or in the inter-organization orchestrator (for the centralized architecture). In any cases, the DOTS client should register itself to its peering DOTS server which provides the intra/inter organization 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 the 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;
            "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;
            "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;
    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, "0" represents all the ports are to be protected;
protected_protocol: Valid protected protocol values include tcp and udp, "all" represents all the protocols are to be protected;
countermeasures: Some of the protection need mitigation and others need Blackholing, "all" represents the DOTS client can accept all kinds of countermeasures;
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 by itself;
next_hop: The returning path to the customer's network. "null" represents there is no next hop information provided and the DOTS server can decide it by 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;
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 ip packet header used in the white-list;
length: The length attribute in ip packet header used in the white-list;
TTL: The TTL attribute in ip packet header used in the white-list;
DSCP: The DSCP attribute in 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;

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 ip packet header used in the black-list;
length: The length attribute in ip packet header used in the black-list;
TTL: The TTL attribute in ip packet header used in the black-list;
DSCP: The DSCP attribute in 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 longed over 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 as 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 cancel the registration;

       registration cancelling response body:
       customer_id: The unique id of the customer (DOTS client);
       result: The final result if the DOTS controller accepts the
       registration cancelling request.
```

5.1.2. Operations

The main operations in the provisioning stage include:

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

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

- 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:

- 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;

- 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 Support;
   5: Security Profile Not Support;
   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-organization 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 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 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;
       "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. 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 DOTS client can undertake.
The unit is "G bytes";
packet_header: IP packet header contents used for report.
CSV (Comma Separated Values) format is used here when multiple values are possible. Note that no spaces between comma's for CSV format, and the multiple values for every attribute should be in the same order as they are assigned.
   dst_ip: A single /32 ip under attack;
   dst_ports: The destination port(s) used for the attack;
   src_ips: The list of source ips of the attack;
   src_ports: The source port(s) used for the attack;

protocols: The protocol numbers used for the attack. The list of IP protocol numbers are defined and maintained by IANA;
tcp_flags: The tcp flags used for the attack. Possible value could be: SYN, FIN, ACK, PSH, RST, URG, NULL;
fragment: The fragment flags in ip header for the attack. Possible value could be: DF - Don't fragment, IsF - Is a fragment, FF - First fragment, LF - Last fragment;
pkt_len: The packet length used for the attack;
icmp_type: The icmp type used for the attack;
icmp_code: The icmp code used for the attack;
DSCP: The DSCP value used for the attack.;
TTL: The TTL value used for the attack;
current_throughputs: Current throughput in bps/pps for the above attack flows
bps: bytes per second;
pps: packets per second;

```
        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;
           pps: packets per second;
        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;
           pps: packets per second;
        info: Other general information which is possibly useful
           attack_types: List of attacks being used together for this
           attack, on this single DST_IP;
           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. 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;
destination_ports: For TCP or UDP or SCTP or DCCP: the destination range of ports (e.g., 1-443) of the discarded traffic;
source_ips: The source IP addresses or prefixes of the discarded traffic;
destination_ip: The destination IP addresses or prefixes of the discarded traffic;
TCP_flags: TCP flag of the discarded traffic;
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 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. 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. 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. 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 malicious attack, requests mitigation service to its operator controller (DOTS server);

- DOTS server authenticates and provides its intra- organization

mitigation service to the DOTS client;

- When the DOTS server are mitigating the attack and finding the attack volume exceeds it capacity, or the attack type is unknown type, or its upstream link is congested, it should request to other DOTS server for inter-organization 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 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 stop its mitigation service and notifies it to DOTS client by sending DOTS status updates message. At last, DOTS client sends a DOTS mitigation termination acknowledgement message to finish the whole DOTS session;

- The heartbeat message is exchange between the DOTS client and 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 they never reached their destination.

6. Other Considerations

6.1. Billing Data

This is not technical nor a part of dots protocol but it has relation to deployment models. If other organization utilized resources of DDoS protection service, it is natural to charge it according to the amount of use. However, how to count the amount of use differs among 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 volume of normal traffic. Number of execution can be also used. We can not decide what information should be taken into account for billing purpose in advance, however those information is needed to be exchanged while coordinating DDoS protection. These information could be also used to determine which service would be used when asking for help. Though it is out of the scope of dots, coordinating and optimizing the cooperation in the aspect of business 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, K. Nishizuka., "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, October 2015".

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.

EMail: Rich.Compton@charter.com