

Tencent Cloud EdgeOne

L4 Proxy

Product Documentation



Copyright Notice

©2013-2025 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by the Tencent corporate group, including its parent, subsidiaries and affiliated companies, as the case may be. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

L4 Proxy

- Overview

- Creating an L4 Proxy Instance

- Modifying an L4 Proxy Instance

- Disabling or Deleting an L4 Proxy Instance

- Batch Configuring Forwarding Rules

- Obtaining Real Client IPs

 - Passing Real TCP Client IPs via TOA

 - Obtaining Real Client IPs Through Protocol V1/V2

 - Overview

 - Method 1: Obtaining Real Client IPs Through Nginx

 - Method 2: Parsing Real Client IPs on Application Server

 - Format of Real Client IPs Obtained Through Proxy Protocol V1/V2

 - Passing Real Client IP Through SPP

L4 Proxy Overview

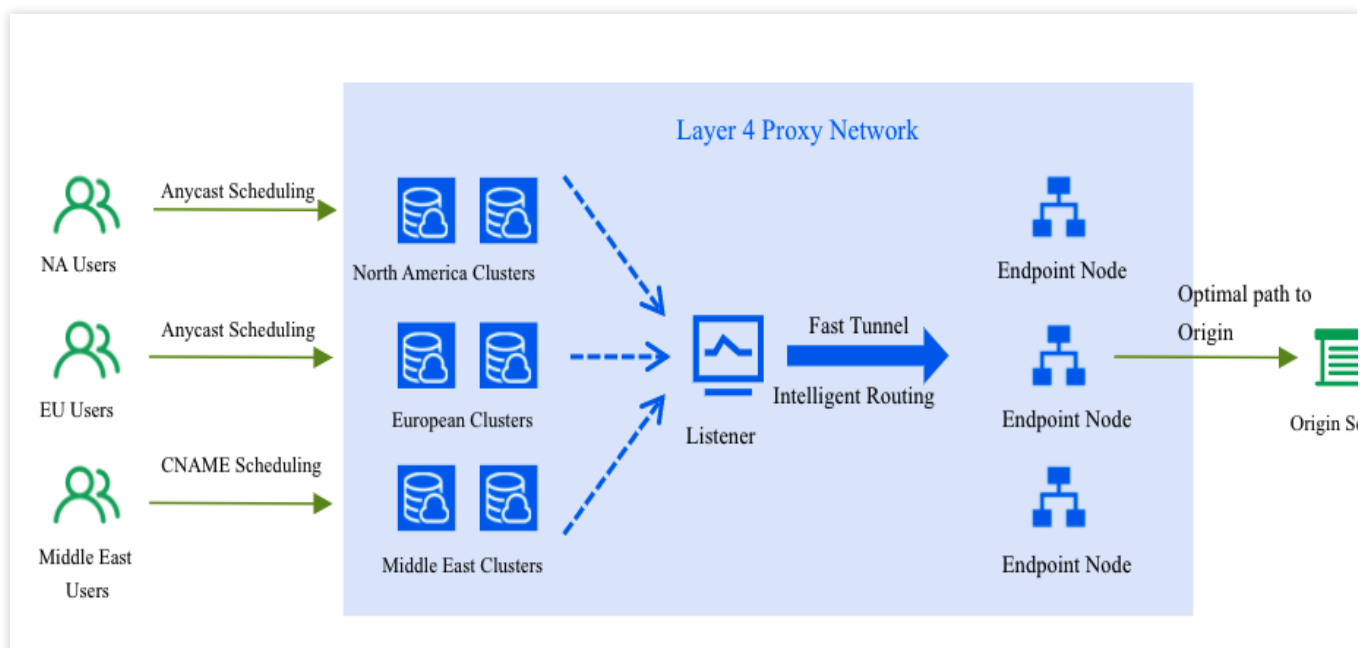
Last updated : 2024-12-25 16:23:59

How It Works

L4 proxy is the acceleration service of EdgeOne based on TCP/UDP. By leveraging widely distributed layer-4 proxy nodes, unique DDoS module, and smart routing technology, EdgeOne implements nearby access for end users, edge traffic cleansing, and port monitoring and forwarding. It thus offers high-availability and low-latency DDoS mitigation and acceleration services for layer-4 applications.

Note:

The L4 proxy is only available with the Enterprise Edition package.



Use Cases

Game Acceleration

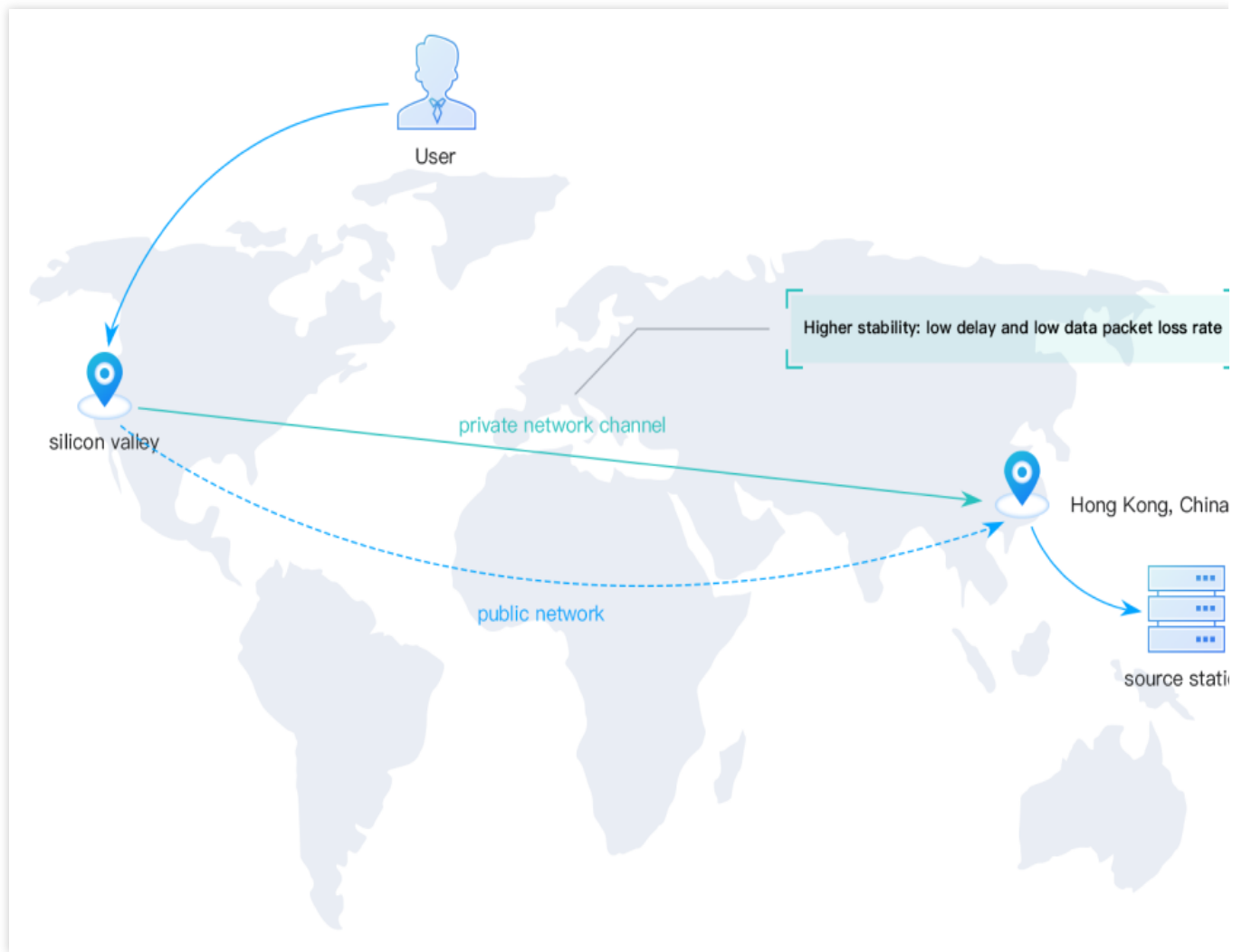
L4 proxy accelerates data transmission over TCP/UDP for mobile and PC games, such as real-time battle games and MMORPGs that require global access to a unified server. L4 proxy connects players to the nearest high-speed

channels to reduce the packet loss rate and latency of the game due to varying network conditions across regions.



OA Application Acceleration

Generally, in cross-regional OA scenarios, the business data of a company is stored in the master data center at its headquarters. This often results in a high packet loss rate with high latency during cross-regional communication due to network issues, causing troubles in cross-regional business access and data synchronization. L4 proxy effectively solves those network issues and improves the business access experience by connecting users to the nearest EdgeOne nodes and optimizing the access links.



Real-time Audio/Video

L4 proxy supports forwarding acceleration over UDP. This ensures reliable audio and video transmission in real-time interactive scenarios, such as video meetings and video communication between anchors and audience members. L4 proxy solves network issues such as audio/video lags, packet loss, and high latency during cross-ISP, long distance, and cross-border communication.

Creating an L4 Proxy Instance

Last updated : 2024-12-25 16:24:21

Use Cases

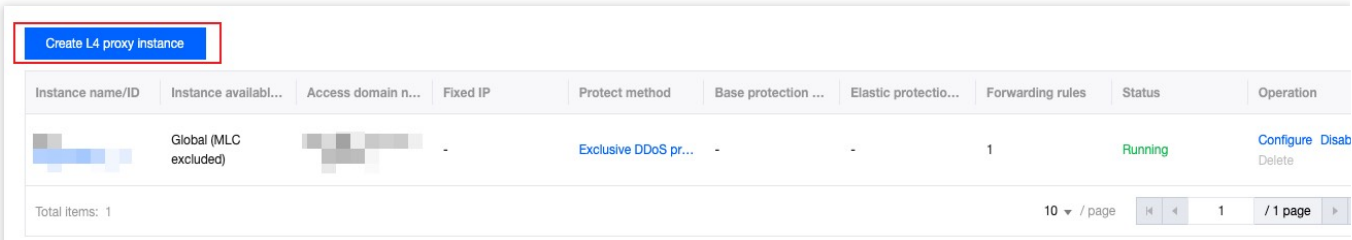
This document describes how to create and configure an L4 proxy instance.

Note:

The L4 proxy is only available with the Enterprise Edition package.

Directions

1. Log in to the [EdgeOne](#) console and click **Site List** in the left sidebar. In the site list, click the target site.
2. On the site details page, click **L4 proxy**.
3. On the page that appears, click **Create L4 proxy instance**.



4. Specify parameters on the **Service Configurations** page. By default, the service region is the accelerated region of the current site. The table below lists the parameters:

Create L4 proxy instance

Instance name

1-50 characters ([a-z], [0-9] and [-]). It must start and end with a digit or letter. Consecutive hyphens (-) are not allowed. After creation, modifications are not allowed.

Instance available area

☒ Global (MLC excluded)

☐ Chinese mainland

☐ Global

Security configuration

Protect method

Default protection

[What is Default protection?](#)

Access configuration

IPv6 access

☐

Cross-MLC-border acceleration

☐

☐ I have read and agree to [EdgeOne Service Level Agreement](#) and [Refund Rule](#)

Subscription fee

/month

Subscribe

Cancel

| Item | Description |
|---------------------------------|---|
| Instance name | 1-50 characters ([a-z], [0-9] and [-]). It must start and end with a digit or letter. Consecutive hyphens (-) are not allowed. After creation, modifications are not allowed. |
| Security Configuration | Default protection: Enabled by default, for details, please refer to DDoS Protection Overview . Exclusive DDoS Protection: For details, please refer to the usage of Exclusive DDoS Protection . |
| IPv6 access | If you enable this feature, EdgeOne nodes can be accessed over the IPv6 protocol. |
| Chinese MLC-border acceleration | When enabled, it will optimize the access performance for Chinese mainland users. For details, please refer to Cross-Regional Secure Acceleration (Overseas Sites) . |

Note:

The Chinese mainland availability zones and global availability zones do not support default protection, but only support exclusive DDoS protection; the global availability zones (excluding the Chinese mainland) support both default protection and exclusive DDoS protection.

5. View subscription fees, check and agree to the [EdgeOne Service Level Agreement](#) and [Refund Policy](#) below, and click Subscribe. For billing description, please refer to the [Billing overview](#).

6. Specify the forwarding rules. On the L4 proxy page, select the newly created L4 proxy instance, click Configuration, enter the instance details page to configure forwarding rules. You can also import multiple forwarding rules at a time. For more information, see [Batch Configuring Forwarding Rules](#). The table below lists the fields of a forwarding rule:

Forwarding rules

Add rule
Batch import
Batch export

| Rule ID | Forwarding... | Forwarding port ⓘ | Origin type ⓘ | Origin address | Origin port ⓘ | Session persistence ⓘ | Pass client IP ⓘ | Rule Tag ⓘ | Status | Operation |
|---------|---------------|-------------------|---------------|----------------|---------------|-----------------------|------------------|------------|--------|-----------|
| - | TCP ▾ | | Origin ▾ | | | No ▾ | TOA ▾ | optional | - | Save Car |

Note:

- If you specify `Origin group` for **Origin type**, you can specify only self-owned origins. In this case, a COS bucket is not supported as the origin.
- You can specify at most 2,000 forwarding rules for each L4 proxy instance.

| Item | Description |
|--------------------------------|---|
| Rule ID | Auto-generated, not supported for modification, unique identifier of the rule. |
| Forwarding protocol | Forwarding protocol of L4 proxy. Valid values: TCP and UDP. |
| Forwarding port | <p>The supported port number ranges from 1 to 64999. You can enter multiple ports separated with semicolons (;) or use a hyphen to enter a port range.</p> <p>The following ports are reserved for internal use, please do not use them:</p> <p>For TCP forwarding protocol: 3943, 3944, 6088, 36000, 56000.</p> <p>For UDP forwarding protocol: 4789, 4790, 6080, 61708.</p> |
| Origin type and Origin address | <p>Single origin: If you specify <code>Single origin</code> for Origin type, you can enter the IP address or domain name of a single origin.</p> <p>Origin group: If you specify <code>Origin group</code> for Origin type, you can select an origin from an existing origin group, or create an origin group.</p> |
| Origin port | <p>You can enter a single port or a port range. If it is a port range, the forwarding port must also be a port range, and the length of the origin port and forwarding port ranges must be consistent.</p> <p>For example: If the forwarding port range is <code>80-90</code>, the origin port range can be <code>80-90</code> or <code>90-100</code>.</p> |
| Session persistence | As long as an origin server IP remains unchanged, traffic from the same client IP will always be forwarded to the same origin server IP. |
| Pass client IP | <p>TOA: Pass client IPs via TCP Option (type 200), which only supports TCP protocols. For more information, see Obtaining Real TCP Client IPs via TOA.</p> <p>Proxy Protocol V1 (recommended): Pass client IPs as plaintext by using the TCP header, which only supports TCP protocols. For more information, see Obtaining Real Client IPs Through Protocol V1/V2.</p> <p>Proxy Protocol V2: Pass client IPs by using the header. V2 uses the binary format and supports both TCP and UDP protocols. The first packet of each TCP connection carries a PPv2 header, while only the first data packet carries the header for UDP. For more information, see Passing Real Client IP Through SPP.</p> |

| | |
|----------|--|
| | Not passed: Real client IPs will not be transferred. |
| Rule Tag | Optional, you can enter 1-50 any characters to identify the forwarding rule. |

7. Click **Save** to complete the configuration of the L4 proxy rules.

Modifying an L4 Proxy Instance

Last updated : 2024-12-25 16:24:38

Use Cases

This document describes how to modify the configuration of an L4 proxy instance.

Note:

The L4 proxy is only available with the Enterprise Edition package.

After an L4 proxy instance is created, you cannot modify its scheduling mode or proxy mode. To do so, you can delete the instance and create a new one.

You can disable and then delete a forwarding rule.

Directions

1. Log in to the [EdgeOne](#) console and click **Site List** in the left sidebar. In the site list, click the target site.
 2. On the site details page, click **L4 proxy**.
 3. On the page that appears, find the target L4 proxy instance and click **configure** in the **Operation** column.
 4. On the page that appears, you can enable or disable IPv6 access and modify Chinese MLC- border acceleration.
- You can also add, edit, enable/disable, or delete a forwarding rules.

Instance configuration

[Disable](#)

Instance ID sid-
Instance name test
Service area Global (MLC excluded)
Access domain name
IPv6 access ☒

Security

Protect method Exclusive DDoS protection

[View protection details](#)

Forwarding rules

[Add rule](#)

[Batch import](#)

[Batch export](#)

| Rule ID | Forwarding... | Forwarding port | Origin type | Origin address | Origin port | Session persistence | Pass client IP | Rule Tag | Status | Operation |
|----------------|---------------|-----------------|-------------|----------------|-------------|---------------------|----------------|----------|---------|---|
| rule-df80dd... | TCP | 80-90 | Origin | | 80-90 | No | TOA | tag test | Running | Edit Suspend Delete |

Disabling or Deleting an L4 Proxy Instance

Last updated : 2024-12-25 16:24:55

Use Cases

This document describes how to disable or delete an L4 proxy instance.

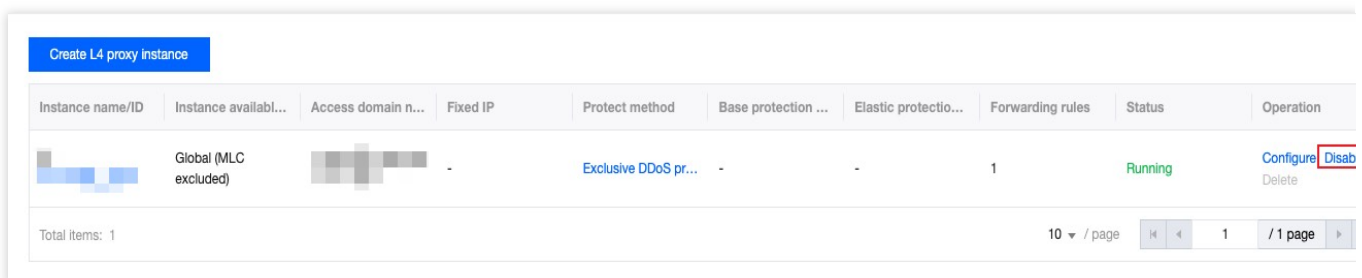
Note:



The L4 proxy is only available with the Enterprise Edition package.

To delete an L4 proxy instance, you must disable it first, which usually takes a few minutes.

Directions

1. Log in to the [EdgeOne](#) console and click **Site List** in the left sidebar. In the site list, click the target site.
2. On the site details page, click **L4 proxy**.
3. On the page that appears, find the target L4 proxy instance and click **Disable** in the **Operation** column.



| Create L4 proxy instance | | | | | | | | | |
|---|-----------------------|---|----------|----------------------|---------------------|----------------------|------------------|-----------|---|
| Instance name/ID | Instance availabl... | Access domain n... | Fixed IP | Protect method | Base protection ... | Elastic protectio... | Forwarding rules | Status | Operation |
|  | Global (MLC excluded) |  | - | Exclusive DDoS pr... | - | - | 1 | Running | Configure Disable |
| Total items: 1 | | | | | | | | 10 / page | 1 / 1 page |

4. Click **Delete** as needed.

Batch Configuring Forwarding Rules

Last updated : 2024-12-25 16:25:12

Use Cases

Tencent Cloud EdgeOne allows you to configure multiple forwarding rules for an L4 proxy instance. This document describes how to import and export multiple forwarding rules at a time.

Note:

1. The L4 proxy is only available with the Enterprise Edition package.
2. You can import up to 2,000 forwarding rules at a time. Each L4 proxy instance supports up to 2,000 forwarding rules.
3. The fields in batch imported forwarding rules are not case-sensitive.
4. The imported forwarding rules cannot use the forwarding ports of existing forwarding rules.

Directions

Importing multiple forwarding rules at a time

1. Log in to the [EdgeOne](#) console and click **Site List** in the left sidebar. In the site list, click the target site.
2. On the site details page, click **L4 proxy**.
3. On the page that appears, find the target L4 proxy instance and click **View** in the **Operation** column.
4. On the page that appears, click **Batch import**.

| Forwarding rules | | | | | | | | | | |
|---|---------------|-------------------|---------------|----------------|---------------|-----------------------|------------------|------------|---------|---|
| <div>Add ruleBatch importBatch export</div> | | | | | | | | | | |
| Rule ID | Forwarding... | Forwarding port ⓘ | Origin type ⓘ | Origin address | Origin port ⓘ | Session persistence ⓘ | Pass client IP ⓘ | Rule Tag ⓘ | Status | Operation |
| | TCP | 80-90 | Origin | | 80-90 | No | TOA | tag test | Running | Edit Suspend Delete |

5. In the pop-up window, enter the forwarding rules to be imported. You must enter one rule per row and specify the forwarding protocol, forwarding port, origin address, origin port, session persistence status, and IP passing mode. Separate fields with spaces. Example: `tcp:123 test.origin.com 456 on ppv1` .

Import forwarding rules in batches



- Enter one forwarding rule per line. You can enter up to 2000 rules
- Each line can have up to 5 fields with case insensitive. Separate them by spaces.
- The fields from left to right are: Forwarding protocol port, origin address, origin port, session persistence status, and IP passing method. [Learn more](#)
- Example: tcp:123 test.origin.com 456 on ppv1

tcp:123 test.origin.com 456 on ppv1

1999 more entries allowed

OK

Cancel

The table below lists the fields of a forwarding rule:

| Field | Description |
|-------------------------------------|--|
| Forwarding protocol:Forwarding port | <p>The supported forwarding protocols are TCP and UDP.</p> <p>The supported port number ranges from 1 to 64999, You can enter multiple ports separated with semicolons (;) or use a hyphen to enter a port range.</p> <p>The following ports are reserved for internal use, please do not use them:</p> <p>For TCP forwarding protocol: 3943, 3944, 6088, 36000, 56000.</p> <p>For UDP forwarding protocol: 4789, 4790, 6080, 61708.</p> |
| Origin address | <p>If you specify <code>Single origin</code> for Origin type, you can enter the IP address or domain name of a single origin.</p> <p>If you specify <code>Origin group</code> for Origin type, you can enter the name of an existing origin group in the format of <code>og:{OriginGroupName}</code> . Example: <code>og:testorigin</code> .</p> |
| Origin port | <p>You can enter a single port or a port range. If it is a port range, the forwarding port must also be a port range, and the length of the origin port and forwarding port ranges must be consistent.</p> |

| | |
|---------------------|--|
| | For example: If the forwarding port range is <code>80-90</code> , the origin port range can be <code>80-90</code> or <code>90-100</code> . |
| Session persistence | Valid values: on and off. |
| Pass client IP | Valid values: toa, ppv1, ppv2, and off. |
| Rule Tag | Optional, you can enter 1-50 any characters to identify the forwarding rule. |

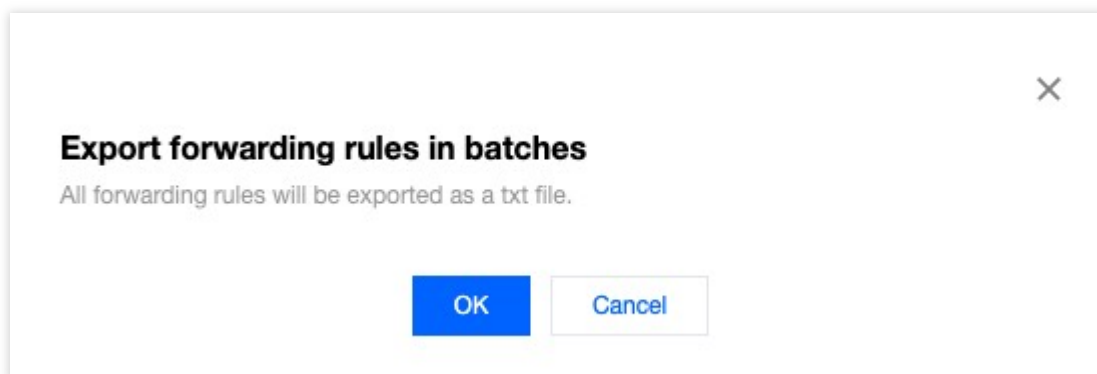
6. Click **OK**.

Exporting multiple forwarding rules at a time

1. Log in to the [EdgeOne](#) console and click **Site List** in the left sidebar. In the site list, click the target site.
2. On the site details page, click **L4 proxy**.
3. On the page that appears, find the target L4 proxy instance and click **View** in the **Operation** column.
4. On the page that appears, click **Batch export**.

| Forwarding rules | | | | | | | | | | |
|---|---------------|-------------------|---------------|----------------|---------------|-----------------------|------------------|------------|---------|---|
| <div>Add rule Batch import Batch export</div> | | | | | | | | | | |
| Rule ID | Forwarding... | Forwarding port ⓘ | Origin type ⓘ | Origin address | Origin port ⓘ | Session persistence ⓘ | Pass client IP ⓘ | Rule Tag ⓘ | Status | Operation |
| | TCP | 80-90 | Origin | | 80-90 | No | TOA | tag test | Running | Edit Suspend Delete |

5. In the pop-up window, click **OK** to export all forwarding rules to a .txt file. The format of the exported rules is the same as that of the imported rules.



Obtaining Real Client IPs

Passing Real TCP Client IPs via TOA

Last updated : 2025-03-10 17:40:14

You can use this document to learn how to get the TCP client IP via TOA when using L4 proxy.

Use Cases

Using L4 acceleration for data packets will have the source IP and port modified, so the origin does not get the original information. To enable the origin to get the real client IP and port, you can pass the information using TOA when creating an acceleration channel. In this way, the real client IP and port are passed into the TCP option field. Meanwhile, you need to install TOA on the origin to get that information.

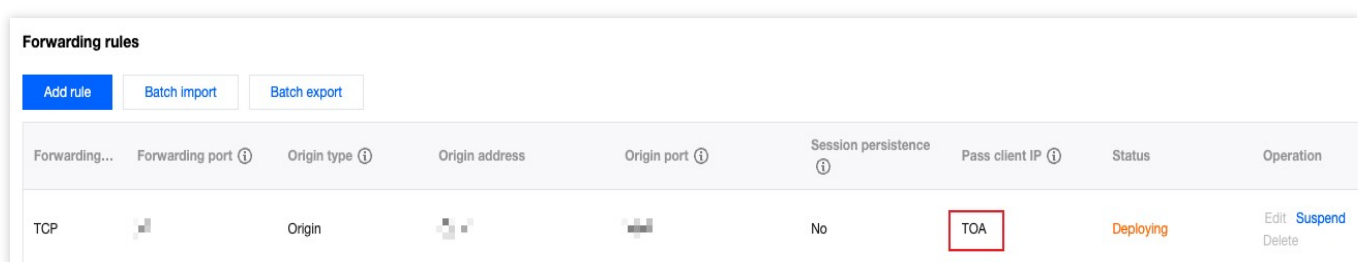
Note:

The L4 proxy is only available with the Enterprise Edition package.

Directions

Step 1: Pass the client IP via TOA

To get the TCP client IP via TOA, set **Pass client IP** to **TOA** in L4 proxy forwarding rules in the console. For details on how to modify rules, see: [Modifying an L4 Proxy Instance](#).



The screenshot shows the 'Forwarding rules' interface in the Tencent Cloud console. At the top, there are buttons for 'Add rule', 'Batch import', and 'Batch export'. Below is a table with columns: Forwarding..., Forwarding port ⓘ, Origin type ⓘ, Origin address, Origin port ⓘ, Session persistence ⓘ, Pass client IP ⓘ, Status, and Operation. A single rule is listed with 'TCP' as the forwarding type, 'Origin' as the origin type, and 'TOA' as the pass client IP method. The status is 'Deploying' and the operation column has links for 'Edit', 'Suspend', and 'Delete'.

| Forwarding... | Forwarding port ⓘ | Origin type ⓘ | Origin address | Origin port ⓘ | Session persistence ⓘ | Pass client IP ⓘ | Status | Operation |
|---------------|-------------------|---------------|----------------|---------------|-----------------------|------------------|-----------|---------------------|
| TCP | | Origin | | | No | TOA | Deploying | Edit Suspend Delete |

Step 2: Load TOA on backend server

You can load the TOA module using either of the following methods:

Method 1 (recommended): Based on the Linux version the origin uses, download the complied toa.ko file and load it directly.

Method 2: If you cannot find the appropriate Linux version, download the TOA source code file and compile and load it yourself. The source code only supports the x86_64 version. If you need support for the arm64 version, please [contact us](#).

Note :

Due to the differences in installation environments, if you encounter issues during the loading process using Method 1, please try Method 2 and install the compilation environment manually.

Method 1: Download and load the compiled TOA module

Method 2: Compile and load the TOA module

1. Download and decompress the TOA package corresponding to the version of Linux OS on Tencent Cloud.

centos

[CentOS-7.2-x86_64.tar.gz](#)

[CentOS-7.3-x86_64.tar.gz](#)

[CentOS-7.4-x86_64.tar.gz](#)

[Centos-7.4-arm64.tar.gz](#)

[CentOS-7.5-x86_64.tar.gz](#)

[CentOS-7.6-x86_64.tar.gz](#)

[CentOS-7.7-x86_64.tar.gz](#)

[CentOS-7.8-x86_64.tar.gz](#)

[CentOS-7.9-x86_64.tar.gz](#)

[Centos7.9-arm64.tar.gz](#)

[CentOS-8.0-x86_64.tar.gz](#)

[CentOS-8.2-x86_64.tar.gz](#)

[Centos8.2-arm64.tar.gz](#)

TencentOS

[TencentOS Server 2.4 for ARM64.tar.gz](#)

[TencentOS Server 3.1 for ARM64.tar.gz](#)

debian

[Debian-12.5-x86_64.tar.gz](#)

[Debian-12.4-x86_64.tar.gz](#)

[Debian-12.0-x86_64.tar.gz](#)

[Debian-11.1-x86_64.tar.gz](#)

[Debian-10.2-x86_64.tar.gz](#)

[Debian-9.0-x86_64.tar.gz](#)

suse linux

[openSUSE-Leap-15.3-x86_64.tar.gz](#)

ubuntu

[Ubuntu-14.04.1-LTS-x86_64.tar.gz](#)

[Ubuntu-16.04.1-LTS-x86_64.tar.gz](#)

[Ubuntu-18.04.1-LTS-x86_64.tar.gz](#)

[Ubuntu18.04-arm64.tar.gz](#)

[Ubuntu-20.04.1-LTS-x86_64.tar.gz](#)

[Ubuntu20.04-arm64.tar.gz](#)

2. After decompression is complete, run the `cd` command to access the decompressed folder. Then load the module as follows:

Load TOA with a script

Manually load TOA

```
/bin/bash -c "$(curl -fsSL https://edgeone-document-file-1258344699.cos.ap-guangzhou.myqcloud.com/TOA/install_toa.sh)"
```

When it is loaded successfully, you will see the following information:

```
[root@VM-0-14-centos toa]# /bin/bash -c "$(curl -fsSL https://eo-toa-1258348367.cos.ap-shanghai.myqcloud.com/install_toa.sh)"
toa.ko install successfully
[root@VM-0-14-centos toa]#
```

```
# Decompress the tar package.
tar -zxvf CentOS-7.2-x86_64.tar.gz
# Enter the directory of the decompressed package.
cd CentOS-7.2-x86_64
# Load the TOA module.
insmod toa.ko
# Copy the TOA module to the kernel module directory.
cp toa.ko /lib/modules/$(uname -r)/kernel/net/netfilter/ipvs/toa.ko
# Configure the TOA module to load automatically at system startup.
echo "insmod /lib/modules/$(uname -r)/kernel/net/netfilter/ipvs/toa.ko" >>
/etc/rc.local
```

Run the following command to check whether the loading is successful:

```
lsmod | grep toa
```

If you see "TOA" in the message, the module is loaded successfully:

```
[root@VM-0-14-centos toa]#
[root@VM-0-14-centos toa]# lsmod | grep toa
toa                282624  0
[root@VM-0-14-centos toa]#
```

1. Install the compilation environment.

1.1 Make sure kernel-devel and kernel-headers are installed and consistent with the kernel version.

1.2 Make sure the gcc and make dependencies are installed.

1.3 If these environmental dependencies are not installed, run the installation command:

CentOS

Ubuntu/Debian

```
yum install -y gcc
yum install -y make
yum install -y kernel-headers kernel-devel

apt-get install -y gcc
apt-get install -y make
apt-get install -y linux-headers-$(uname -r)
```

2. After the compilation environment is installed, download, compile and load the source code.

Compile and load TOA with a script

Manually compile and load TOA

```
/bin/bash -c "$(curl -fsSL https://edgeone-document-file-1258344699.cos.ap-
guangzhou.myqcloud.com/TOA/compile_install_toa.sh) "
```

```
# Create a compilation directory and enter it.
mkdir toa_compile && cd toa_compile
# Download the source code (tar.gz)
curl -o toa.tar.gz https://edgeone-document-file-1258344699.cos.ap-
guangzhou.myqcloud.com/TOA/toa.tar.gz
# Decompress the tar package
tar -zxvf toa.tar.gz
# Compile the toa.ko file. After the compilation is successful, the file will
be generated in the current directory.
make
# Load the TOA module.
insmod toa.ko
# Copy the TOA module to the kernel module directory.
cp toa.ko /lib/modules/$(uname -r)/kernel/net/netfilter/ipvs/toa.ko
# Configure the TOA module to load automatically at system startup
echo "insmod /lib/modules/$(uname -r)/kernel/net/netfilter/ipvs/toa.ko" >>
/etc/rc.local
```

3. Run the following command to check whether the loading is successful:

```
lsmod | grep toa
```

If you see "TOA" in the message, the module is loaded successfully:

```
[root@VM-16-42-centos ~]# lsmod | grep toa
toa                278528  0
```

Step 3: Verify the configuration

You can verify the configuration by building a TCP server to receive client requests from another server. See the sample:

1. On the current server, create an HTTP server in Python to act as a TCP server:

```
# Use python2
python2 -m SimpleHTTPServer 10000

# Use python3
python3 -m http.server 10000
```

2. Make another server work as a client to send requests:

```
# Use curl to initiate an HTTP request, where the hostname and forwarding port
of the L4 proxy is used.
curl -i "http://a8b7f59fc8d7e6c9.example.com.edgeoneddy1.com:10000/"
```

3. If TOA is loaded, the real client address can be seen on the server:

```
[root@VM-0-14-centos tmp]# python2 -m SimpleHTTPServer 10000
Serving HTTP on 0.0.0.0 port 10000 ...
119.29.135.205 - - [26/Apr/2023 17:52:37] "GET / HTTP/1.1" 200 -
```

You can also get either the IPv4 or IPv6 address of the client by following the steps above.

For origin IPv4 addresses, get the client IPv4 address.

For origin IPv6 addresses, get the client IPv6 address.

If you need to get both IPv4 and IPv6 addresses, modify the origin's business code while loading the TOA module as instructed [here](#).

Getting Both IPv4 and IPv6 Addresses

Note:

This section provide guidance on how to get both IPv4 and IPv6 addresses by modifying the business code of the origin.

The origin can listen on requests in either of the following methods:

1. Use the structure `struct sockaddr_in` to listen on IPv4 addresses.
2. Use the structure `struct sockaddr_in6` to listen on IPv6 addresses.

Sample code

[Listen on IPv4 addresses](#)

[Listen on IPv6 addresses](#)

C

Java

```
#include <sys/socket.h>
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <memory.h>
#include <arpa/inet.h>

int main(int argc, char** argv) {
    int l_sockfd;
    // The server address is an IPv4 address.
    struct sockaddr_in serveraddr;
    // In this case, the client address must adopt the IPv6 structure.
    struct sockaddr_in6 clientAddr;
    int server_port = 10000;

    memset(&serveraddr, 0, sizeof(serveraddr));
    // Create a socket.
    l_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (l_sockfd == -1){
        printf("Failed to create socket.\n");
        return -1;
    }

    // Initialize the server.
    memset(&serveraddr, 0, sizeof(struct sockaddr_in));
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_port = htons(server_port);
    serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);

    int isReuse = 1;
    setsockopt(l_sockfd, SOL_SOCKET, SO_REUSEADDR, (const char*)&isReuse, sizeof(i

    // Associate the socket and server address.
    int nRet = bind(l_sockfd, (struct sockaddr*)&serveraddr, sizeof(serveraddr))
    if(-1 == nRet)
    {
        printf("bind error\n");
        return -1;
    }
    // Listen on the socket.
```

```
listen(l_sockfd, 5);

int clientAddrLen = sizeof(clientAddr);
memset(&clientAddr, 0, sizeof(clientAddr));
// Accept connections from the client.
int linkFd = accept(l_sockfd, (struct sockaddr*)&clientAddr, &clientAddrLen);
if(-1 == linkFd)
{
    printf("accept error\\n");
    return -1;
}
// Modifications to make: Decide whether the client is an IPv4 or IPv6 address
// AF_INET indicates that the client adopts IPv4. In this case, convert the
// AF_INET6 indicates that the client adopts IPv6. In this case, use struct
if (clientAddr.sin6_family == AF_INET) {
    printf("AF_INET accept getpeername %s : %d successful\\n",
        inet_ntoa(((struct sockaddr_in*)&clientAddr)->sin_addr),
        ntohs(((struct sockaddr_in*)&clientAddr)->sin_port));
}else if (clientAddr.sin6_family == AF_INET6){
    char addr_p[128] = {0};
    inet_ntop(AF_INET6, (void *)&((struct sockaddr_in6*)&clientAddr)->sin6_addr,
        addr_p,
        ntohs(((struct sockaddr_in6*)&clientAddr)->sin6_port));
}else{
    printf("unknow sin_family:%d \\n", clientAddr.sin6_family);
}
close(l_sockfd);
return 0;
}

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketAddress;

public class ServerDemo {

    /** If using the IPv4 address structure to build the service, use IPV4_HOST */
    public static final String IPV4_HOST = "0.0.0.0";
```

```
/** If using the IPv6 address structure to build the service, use IPV6_HOST */
public static final String IPV6_HOST = "::";

public static void main(String[] args) {
    int serverPort = 10000;
    try (ServerSocket serverSocket = new ServerSocket()) {
        // Setting address reuse
        serverSocket.setReuseAddress(true);
        // Bound server address and port, using IPv4 here
        serverSocket.bind(new InetSocketAddress(InetAddress.getByName(IPV4_HOST)
        System.out.println("Server is listening on port " + serverPort);

        while (true) {
            // Accepting Client connections
            Socket clientSocket = serverSocket.accept();
            System.out.println("New client connected: " + clientSocket.getRemot

            // Processing Client requests
            handleClientRequest(clientSocket);
        }
    } catch (IOException e) {
        System.err.println("Failed to create server socket: " + e.getMessage())
    }
}

/**
 * Processing Function, site business implement, here is just an example
 * The purpose of this Function is to Return the Client's input verbatim to the
 */
private static void handleClientRequest(Socket clientSocket) {
    try (InputStream inputStream = clientSocket.getInputStream();
        OutputStream outputStream = clientSocket.getOutputStream()) {

        // Reading the Data received from the Client
        byte[] buffer = new byte[1024];
        int bytesRead;
        while ((bytesRead = inputStream.read(buffer)) != -1) {
            // Reply the received Data to the Client as it is
            outputStream.write(buffer, 0, bytesRead);
        }
    }
}
```



```
    } catch (IOException e) {
        // When the Client disconnects
        System.err.println("Failed to handle client request: " + e.getMessage())
    } finally {
        try {
            clientSocket.close();
        } catch (IOException e) {
            System.err.println("Failed to close client socket: " + e.getMessage())
        }
    }
}
```

C

Java

```
#include <sys/socket.h>
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <memory.h>
#include <arpa/inet.h>

int main(int argc, char **argv)
{
    int l_sockfd;
    // The server address is an IPv6 address.
    struct sockaddr_in6 serveraddr;
    // The client address is an IPv6 address.
    struct sockaddr_in6 clientAddr;
    int server_port = 10000;

    memset(&serveraddr, 0, sizeof(serveraddr));

    // Create a socket.
    l_sockfd = socket(AF_INET6, SOCK_STREAM, 0);
    if (l_sockfd == -1){
        printf("Failed to create socket.\n");
        return -1;
    }
    // Set the server address.
    memset(&serveraddr, 0, sizeof(struct sockaddr_in6));
    serveraddr.sin6_family = AF_INET6;
    serveraddr.sin6_port = htons(server_port);
    serveraddr.sin6_addr = in6addr_any;
```

```
    int isReuse = 1;
    setsockopt(l_sockfd, SOL_SOCKET, SO_REUSEADDR, (const char*)&isReuse, sizeof(i
// Associate the socket and server address.
    int nRet = bind(l_sockfd, (struct sockaddr*)&serveraddr, sizeof(serveraddr))
    if(-1 == nRet)
    {
        printf("bind error\\n");
        return -1;
    }
// Listen on the socket.
    listen(l_sockfd, 5);

    int clientAddrLen = sizeof(clientAddr);
memset(&clientAddr, 0, sizeof(clientAddr));

// Accept connection requests from the client.
int linkFd = accept(l_sockfd, (struct sockaddr*)&clientAddr, &clientAddrLen);
if(-1 == linkFd)
{
    printf("accept error\\n");
    return -1;
}

// The client addresses received here are all stored in the IPv6 structure.
// The IPv4 addresses are mapped to IPv6 addresses, for example, "::ffff:119.29
char addr_p[128] = {0};
inet_ntop(AF_INET6, (void *)&clientAddr.sin6_addr, addr_p, (socklen_t )sizeof(a
printf("accept %s : %d successful\\n", addr_p, ntohs(clientAddr.sin6_port));

// Modifications to make: Use the macro definition IN6_IS_ADDR_V4MAPPED to deci
if(IN6_IS_ADDR_V4MAPPED(&clientAddr.sin6_addr)) {
    struct sockaddr_in real_v4_sin;
    memset (&real_v4_sin, 0, sizeof (struct sockaddr_in));
    real_v4_sin.sin_family = AF_INET;
    real_v4_sin.sin_port = clientAddr.sin6_port;
    // The last four bytes represent the IPv4 address of the client.
    memcpy (&real_v4_sin.sin_addr, ((char *)&clientAddr.sin6_addr) + 12, 4);
    printf("connect %s successful\\n", inet_ntoa(real_v4_sin.sin_addr));
}

    close(l_sockfd);
    return 0;
}

import java.io.IOException;
```

```
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketAddress;

public class ServerDemo {

    /** If using the IPv4 address structure to build the service, use IPV4_HOST */
    public static final String IPV4_HOST = "0.0.0.0";

    /** If using the IPv6 address structure to build the service, use IPV6_HOST */
    public static final String IPV6_HOST = "::";

    public static void main(String[] args) {
        int serverPort = 10000;
        try (ServerSocket serverSocket = new ServerSocket()) {
            // Setting address reuse
            serverSocket.setReuseAddress(true);
            // Bound server address and port, using IPv4 here
            serverSocket.bind(new InetSocketAddress(InetAddress.getByName(IPV6_HOST), serverPort));
            System.out.println("Server is listening on port " + serverPort);

            while (true) {
                // Accepting Client connections
                Socket clientSocket = serverSocket.accept();
                System.out.println("New client connected: " + clientSocket.getRemoteAddress());

                // Processing Client requests
                handleClientRequest(clientSocket);
            }
        } catch (IOException e) {
            System.err.println("Failed to create server socket: " + e.getMessage());
        }
    }

    /**
     * Processing Function, site business implement, here is just an example
     * The purpose of this Function is to Return the Client's input verbatim to the client
     */
    private static void handleClientRequest(Socket clientSocket) {
        try (InputStream inputStream = clientSocket.getInputStream();
            OutputStream outputStream = clientSocket.getOutputStream()) {

            // Reading the Data received from the Client
        }
    }
}
```

```
        byte[] buffer = new byte[1024];
        int bytesRead;
        while ((bytesRead = inputStream.read(buffer)) != -1) {
            // Reply the received Data to the Client as it is
            outputStream.write(buffer, 0, bytesRead);
        }

    } catch (IOException e) {
        // When the Client disconnects
        System.err.println("Failed to handle client request: " + e.getMessage());
    } finally {
        try {
            clientSocket.close();
        } catch (IOException e) {
            System.err.println("Failed to close client socket: " + e.getMessage());
        }
    }
}
}
```

console output result

```
Server is listening on port 10000
New client connected: /127.0.0.1:50680
New client connected: /0:0:0:0:0:0:0:1:51124
New client connected: /127.0.0.1:51136
```

References

Monitoring TOA Running Status

To ensure execution stability, this kernel module allows you to monitor status. After inserting the `toa.ko` kernel module, you can monitor the TOA working status in either of the following ways.

```
cat /proc/net/toa_stats
```

This figure shows the TOA running status:

```
[root@VM-16-42-centos ~]# cat /proc/net/toa_stats
              CPU0      CPU1
syn_recv_sock_toa      :      865      858
syn_recv_sock_no_toa   :     1011     1035
getname_toa_ok         :         0         0
getname_toa_mismatch   :      831      892
getname_toa_bypass     :         0         0
getname_toa_empty      :    12897    12757
ip6_address_alloc      :      865      858
ip6_address_free       :      819      904
```

The monitoring metrics are described as follows:

| Metric | Description |
|--|--|
| syn_recv_sock_toa | Receives connections with TOA information. |
| syn_recv_sock_no_toa | Receives connections without TOA information. |
| getname_toa_ok | This count increases when you call <code>getsockopt</code> and obtain the source IP address successfully or when you call <code>accept</code> to receive client requests. |
| getname_toa_mismatch | This count increases when you call <code>getsockopt</code> and obtain a source IP address that does not match the required type. For example, if a client connection contains a source IPv4 address whereas you obtain an IPv6 address, the count will increase. |
| getname_toa_empty | This count increases when the <code>getsockopt</code> function is called in a client file descriptor that does not contain TOA. |
| ip6_address_alloc audio/video proxy | Allocates space to store the information when TOA obtains the source IP address and source port saved in the TCP data packet. |
| ip6_address_free audio/video proxy | When the connection is released, TOA will release the memory previously used to save the source IP and source port. If all connections are closed, the total count of <code>ip6_address_alloc</code> for each CPU should be equal to the count of this metric. |

Obtaining Real Client IPs Through Protocol V1/V2

Overview

Last updated : 2024-12-25 16:26:54

This document describes how to obtain real client IPs through Proxy Protocol V1/V2 when you enable the L4 proxy acceleration.

Note:

The L4 proxy is only available with the Enterprise Edition package.

Scenarios

When the datagrams are accelerated through L4 acceleration connection, you can pass the real client IPs and Ports to the origin server through Proxy Protocol V1/V2. For introduction on the protocol, see [Proxy Protocol V1/V2](#).

The origin can parse and obtain real client IPs with two methods based on the scenarios and deployment modes.

Method 1: If the TCP protocol is used on the origin, it is recommended to add a Nginx server that supports Proxy Protocol V1/V2 in front of the application server. For details, see [Obtaining Real Client IPs Through Nginx](#).

Method 2: If the UDP protocol is used on the origin, or if you want to directly parse the real client IPs under the TCP protocol on the application server, you can parse the Proxy Protocol field on the application server by referring to the sample code in the Proxy Protocol. For details, see [Parsing Real Client IPs on Application Server](#).

Method 1: Obtaining Real Client IPs Through Nginx

Last updated : 2024-12-25 16:27:13

Overview

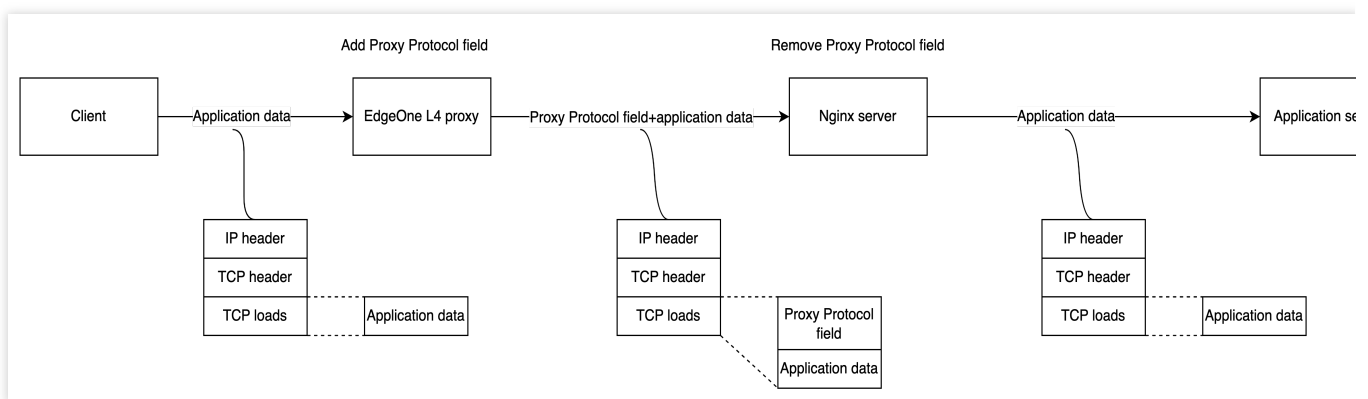
If the TCP protocol is used on the origin, it is recommended to add a Nginx server that supports Proxy Protocol V1/V2 in front of the application server to obtain real client IPs.

Note:

The L4 proxy is only available with the Enterprise Edition package.

If the TCP protocol is used on the origin, and you want to directly parse the real client IPs on the application server, please see [Parsing Real Client IPs on Application Server](#).

Deployment Mode



As shown in the above diagram, you need to deploy a Nginx server in front of the application server to remove the Proxy Protocol field. You can collect the real client IPs by analyzing Nginx logs on the Nginx server. At this time, you can point the origin address to the Nginx service when you configure the origin address in the EdgeOne L4 proxy service.

Directions

Step 1. Deploy Nginx service

Please select a Nginx version corresponding to the Proxy Protocol version you want to use.

For Proxy Protocol V1: Nginx Plus R11 and later versions, Nginx Open Source 1.11.4 and later versions.

For Proxy Protocol V2: Nginx Plus R16 and later versions, Nginx Open Source 1.13.11 and later versions.

For other Nginx versions, see [Accepting the PROXY Protocol](#).

You need to install Nginx-1.18.0 and the stream module to enable L4 proxy service on Nginx. See installation directions below.

```
# Install the nginx build environment
yum -y install gcc gcc-c++ autoconf automake
yum -y install zlib zlib-devel openssl openssl-devel pcre-devel

# Decompress the source package
tar -zxvf nginx-1.18.0.tar.gz
# Enter the directory
cd nginx-1.18.0
# Set nginx compilation and installation configuration (with `--with-stream`)
./configure --prefix=/opt/nginx --sbin-path=/opt/nginx/sbin/nginx --conf-
path=/opt/nginx/conf/nginx.conf --with-http_stub_status_module --with-
http_gzip_static_module --with-stream
# Compilation
make
# Installation
make install
```

Step 2: Configure the stream module in Nginx

If you select Nginx-1.18.0, you can run the following command to open the configuration file nginx.conf.

```
vi /opt/nginx/conf/nginx.conf
```

Configuration of the stream module is as follows:

```
stream {
    # Set the log format, where `proxy_protocol_addr` is the client address obtained
    log_format basic '$proxy_protocol_addr - $remote_addr [$time_local] '
                    '$protocol $bytes_sent $bytes_received '
                    '$session_time';

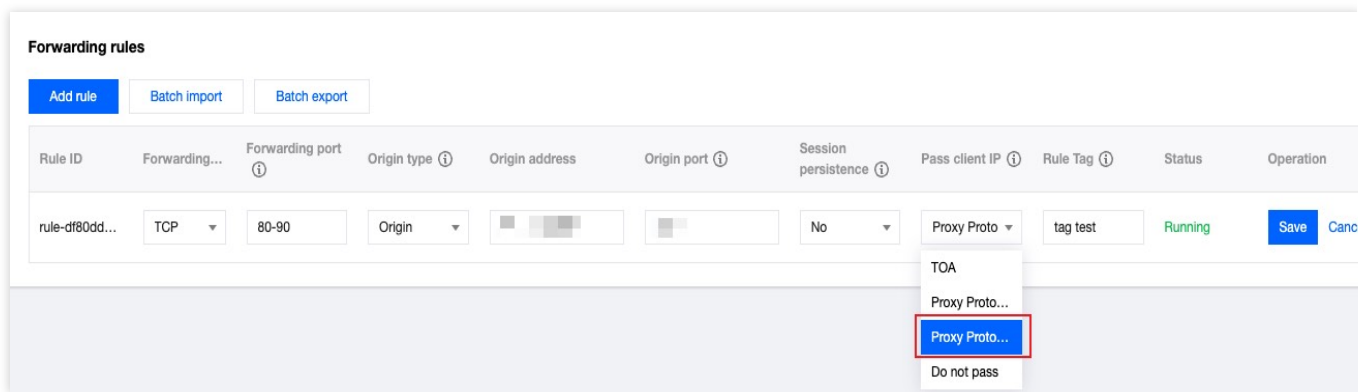
    access_log logs/stream.access.log basic;
    # upstream configuration
    upstream RealServer {
        hash $remote_addr consistent;
        # 127.0.0.1:8888 is the IP address and port of the application server
        server 127.0.0.1:8888 max_fails=3 fail_timeout=30s;
    }
    # server configuration
    server {
```



```
# L4 listening port, which corresponds to the origin port configured in L4
listen 10000 proxy_protocol;
proxy_connect_timeout 1s;
proxy_timeout 3s;
proxy_pass RealServer;
}
}
```

Step 3: Configure L4 proxy forwarding rule

After configuring the Nginx service, you can modify the L4 proxy forwarding rule in the console. Change the origin address to the IP of the current Nginx service, and change the origin port to the L4 listening port configured in [step 2](#). Select Proxy Protocol V1 or V2 for the Pass Client IP according to the forwarding protocol. For details, see [Modifying L4 Proxy Forwarding Rules](#).



Step 4: Simulate client requests and verify results

You can build the TCP service, and simulate client requests on another server to verify the results. A sample is as below:

1. Create an HTTP service with Python on the current server to simulate the TCP service.

```
# Based on python2
python2 -m SimpleHTTPServer 8888

# Based on python3
python3 -m http.server 8888
```

2. Build a client request on another server, and simulate the TCP request with a curl request.

```
# Initiate an HTTP request with curl, where the domain is the L4 proxy domain,
and `8888` is the L4 proxy forwarding port
curl -i "http://d42f15b7a9b47488.davidjli.xyz.acc.edgeoned1.com:8888/"
```

3. Check Nginx logs on the Nginx server:

```

Client IP
119.29.135.205 -43.132.85.50 [28/Apr/2023:15:19:59 +0800] TCP 3 3 3.00

```

You can capture packets on the Nginx server and analyze the packets with Wireshark. After the TCP handshake is completed, the Proxy Protocol field is added in front of the first application data packet. Below is an example for Proxy Protocol V1. ① refers to the L4 proxy egress IP, ② refers to the Nginx server IP, ③ refers to the protocol version, ④ refers to the real client IP.

| | | | | |
|----|-----------|----------------|--------------|---------|
| 17 | 5.887806 | 43.132.85.50 | 10.4.0.14 | TCP |
| 18 | 8.271624 | 43.132.85.50 ① | 10.4.0.14 ② | PROXYv1 |
| 19 | 8.271703 | 127.0.0.1 | 127.0.0.1 | TCP |
| 20 | 8.271749 | 127.0.0.1 | 127.0.0.1 | TCP |
| 21 | 8.271755 | 127.0.0.1 | 127.0.0.1 | TCP |
| 22 | 8.271820 | 10.4.0.14 | 43.132.85.50 | TCP |
| 23 | 8.408399 | 43.132.85.50 | 10.4.0.14 | TCP |
| 24 | 10.927932 | 43.132.85.50 | 10.4.0.14 | TCP |
| 25 | 10.927994 | 127.0.0.1 | 127.0.0.1 | TCP |
| 26 | 10.928045 | 127.0.0.1 | 127.0.0.1 | TCP |
| 27 | 10.928051 | 127.0.0.1 | 127.0.0.1 | TCP |

Frame 18: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)
Linux cooked capture v1
Internet Protocol Version 4, Src: 43.132.85.50, Dst: 10.4.0.14
Transmission Control Protocol, Src Port: 7502, Dst Port: 10000, Seq: 57, Ack: 4, Len: 4
[2 Reassembled TCP Segments (60 bytes): #7(56), #18(4)]

PROXY Protocol

PROXY v1 magic
Protocol: TCP4
Source Address: 119.29.135.205 ④
Destination Address: 43.159.115.63
Source Port: 53859
Destination Port: 10000
Data (7 bytes)

Method 2: Parsing Real Client IPs on Application Server

Last updated : 2024-12-25 16:27:34

Scenarios

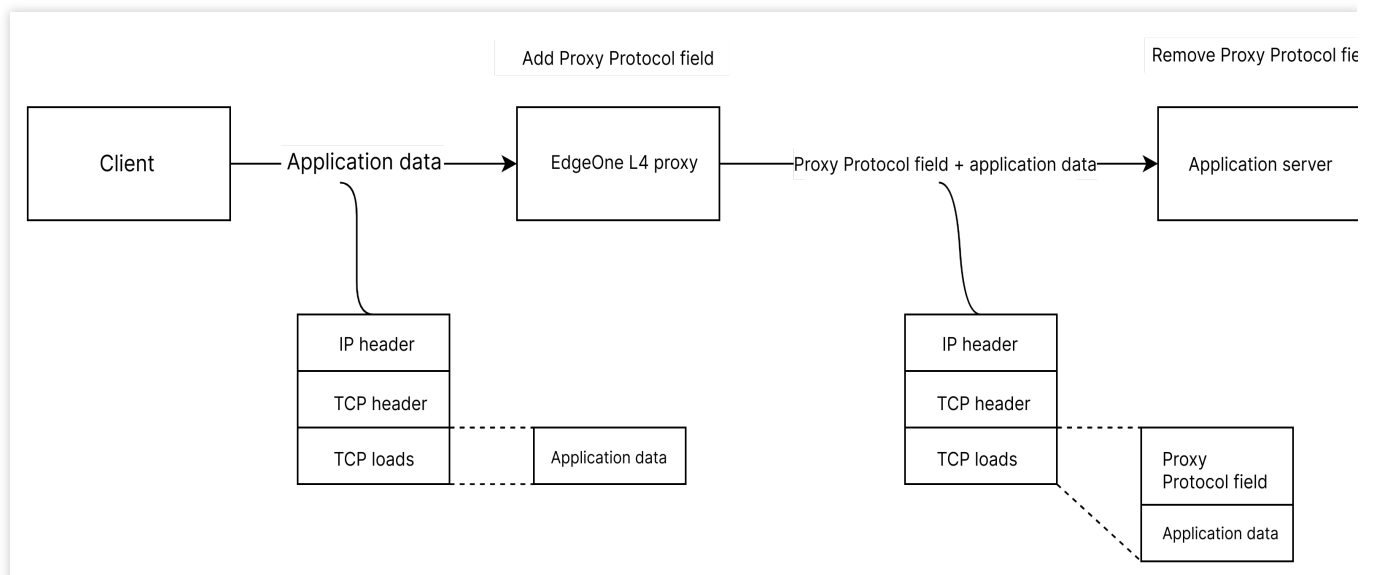
Scenario 1: If the UDP protocol is used on the origin, only Proxy Protocol V2 can be selected to pass the real client IPs. In this case, you need to parse the Proxy Protocol V2 on the application server to obtain the real client IPs.

Scenario 2: If the TCP protocol is used on the origin, and you want to implement application judgment via the real client IPs on the application server, you need to parse the Proxy Protocol V1/V2 on the application server to obtain the real client IPs.

Note:

The L4 proxy is only available with the Enterprise Edition package.

Deployment Diagram



As shown in the above diagram, you can configure L4 proxy via EdgeOne L4 proxy module to point to the application server, and add the Proxy Protocol field to the application data by EdgeOne L4 proxy service. Parsing is implemented on the application server.

Directions

Step 1: Configure L4 proxy forwarding rule

Modify the L4 proxy forwarding rule in the console. You need to enter the origin address and origin port. If the forwarding protocol is UDP, select Proxy Protocol V2 for Pass client IP. If the forwarding protocol is TCP, you can select Proxy Protocol V1 or V2. For details, see [Modifying L4 Proxy Forwarding Rules](#).

| Rule ID | Forwarding... | Forwarding port | Origin type | Origin address | Origin port | Session persistence | Pass client IP | Rule Tag | Status | Operation |
|----------------|---------------|-----------------|-------------|----------------|-------------|---------------------|----------------|----------|---------|---|
| rule-df80dd... | UDP | 80-90 | Origin | | | No | Proxy Proto | tag test | Running | Save Cancel |

Step 2: Obtain real client IPs on the application server

You need to parse the Proxy Protocol filed with reference to the [sample code](#) in the [Proxy Protocol](#). For the format of the client IPs, see [Format of Real Client IPs Obtained Through Proxy Protocol V1/V2](#).

When the UDP protocol and Proxy Protocol V2 are selected, the Proxy Protocol field is added to the first UDP datagram. In the figure below, ① refers to the L4 proxy egress IP, ② refers to the origin address, ③ refers to the protocol version, ④ refers to the Proxy Protocol field, ⑤ refers to the real client IP address, and ⑥ refers to the application data.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-----------|----------------|--------------|-----------|--------|-------------------|
| 1 | 0.000000 | 43.175.17.39 ① | 10.4.0.14 ② | PROXYv2 ③ | 73 | 11834 → 38888 Len |
| 4 | 0.000205 | 10.4.0.14 | 43.175.17.39 | UDP | 81 | 38888 → 11834 Len |
| 5 | 2.230466 | 43.175.17.39 | 10.4.0.14 | UDP | 45 | 11834 → 38888 Len |
| 8 | 2.230619 | 10.4.0.14 | 43.175.17.39 | UDP | 53 | 38888 → 11834 Len |
| 9 | 6.235155 | 43.175.17.39 | 10.4.0.14 | UDP | 45 | 11834 → 38888 Len |
| 12 | 6.235324 | 10.4.0.14 | 43.175.17.39 | UDP | 53 | 38888 → 11834 Len |
| 13 | 8.466705 | 43.175.17.39 | 10.4.0.14 | UDP | 45 | 11834 → 38888 Len |
| 16 | 8.466900 | 10.4.0.14 | 43.175.17.39 | UDP | 53 | 38888 → 11834 Len |
| 17 | 10.697625 | 43.175.17.39 | 10.4.0.14 | UDP | 45 | 11834 → 38888 Len |
| 20 | 10.697773 | 10.4.0.14 | 43.175.17.39 | UDP | 53 | 38888 → 11834 Len |

> Frame 1: 73 bytes on wire (584 bits), 73 bytes captured (584 bits)

> Linux cooked capture v1

> Internet Protocol Version 4, Src: 43.175.17.39, Dst: 10.4.0.14

> User Datagram Protocol, Src Port: 11834, Dst Port: 38888

▼ PROXY Protocol

Magic: 0d0a0d0a00d0a515549540a ④
0010 = Version: 2
.... 0001 = Command: 1
[Version: 2]
> Address Family Protocol: UDP over IPv4 (0x12)
Length: 12
Source Address: 119.29.135.205
Destination Address: 43.159.115.63 ⑤
Source Port: 48748
Destination Port: 38888

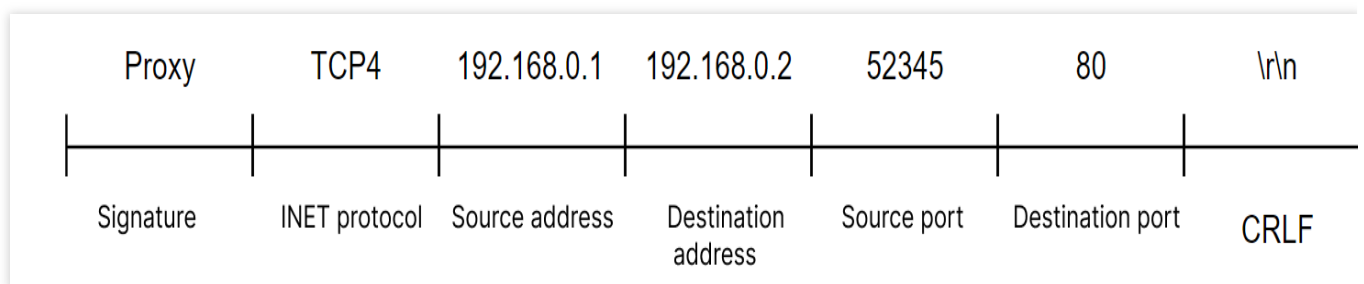
| | | | |
|------|-------------------------|-------------------------|------------------|
| 0000 | 00 00 00 01 00 06 fe ee | 35 c9 48 c9 00 00 08 00 | 5:H..... |
| 0010 | 45 b8 00 39 d7 79 40 00 | 30 11 2b 9b 2b af 11 27 | E..9.y@.0-++.. |
| 0020 | 0a 04 00 0a 2e 3a 97 e8 | 00 25 df 96 0d 0a 0d 0a |:.. %-..... |
| 0030 | 00 0d 0a 51 55 49 54 0a | 21 12 00 0c 77 1d 87 cd | ...QUIT !...w... |
| 0040 | 2b 9f 73 3f be 6c 97 e8 | 30 ⑥ | +..s?.1.. 0 |

Format of Real Client IPs Obtained Through Proxy Protocol V1/V2

Last updated : 2023-06-29 15:37:55

Proxy Protocol V1

Proxy Protocol V1 supports TCPv4 and TCPv6, and adopts string format. See details below:



```
PROXY TCP4 192.168.0.1 192.168.0.11 56324 443\r\n
```

You can check the following information with Wireshark.

▼ PROXY Protocol

PROXY v1 magic

Protocol: TCP4

Source Address: 119.29.135.205

Destination Address: 43.159.115.63

Source Port: 53859

Destination Port: 10000

› Data (7 bytes)

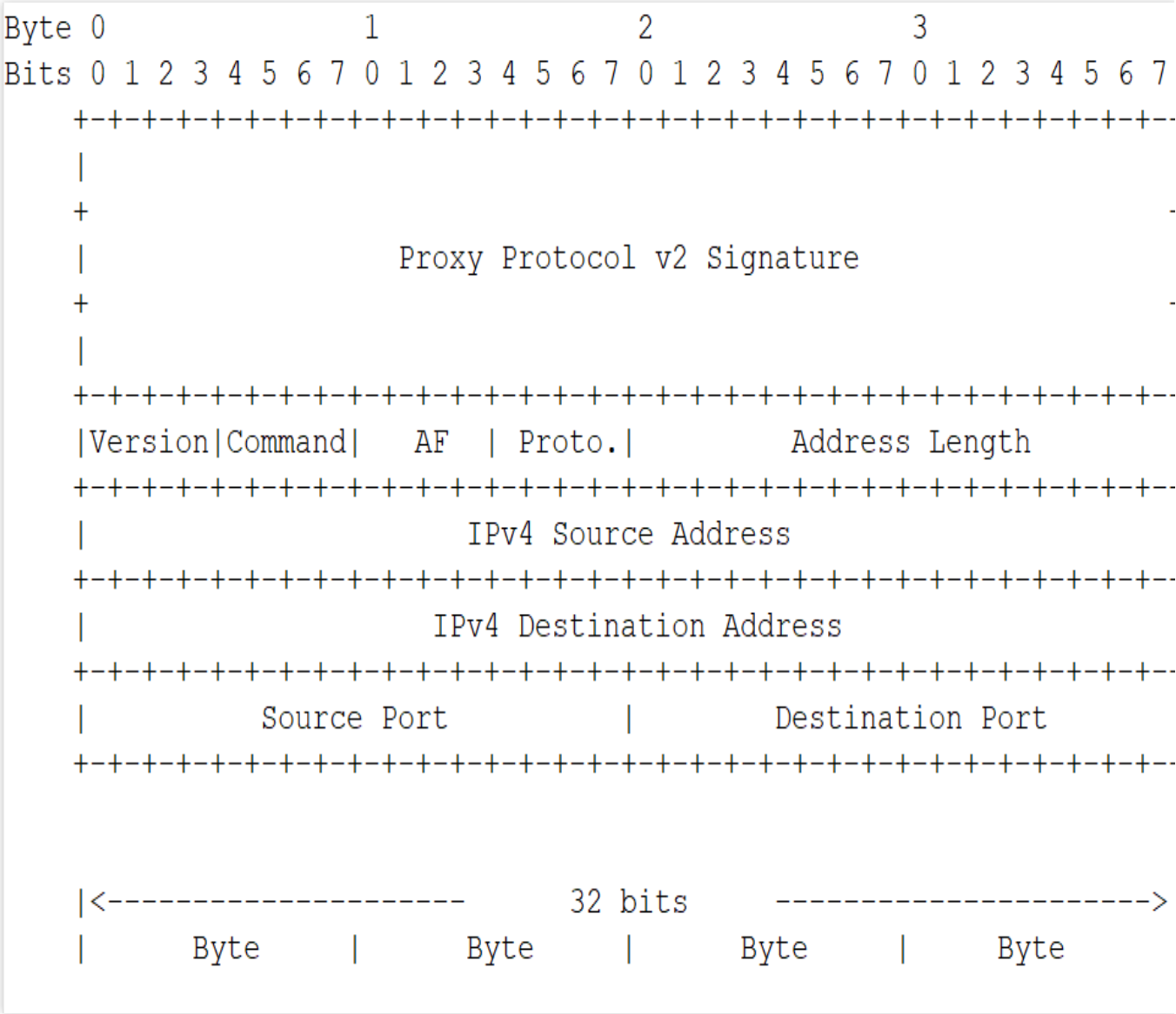
String format

| | | |
|------|---|-------------------|
| 0000 | 50 52 4f 58 59 20 54 43 50 34 20 31 31 39 2e 32 | PROXY TC P4 119.2 |
| 0010 | 39 2e 31 33 35 2e 32 30 35 20 34 33 2e 31 35 39 | 9.135.20 5 43.159 |
| 0020 | 2e 31 31 35 2e 36 33 20 35 33 38 35 39 20 31 30 | .115.63 53859 10 |
| 0030 | 30 30 30 0d 0a 61 62 63 31 32 34 33 | 000··abc 1243 |

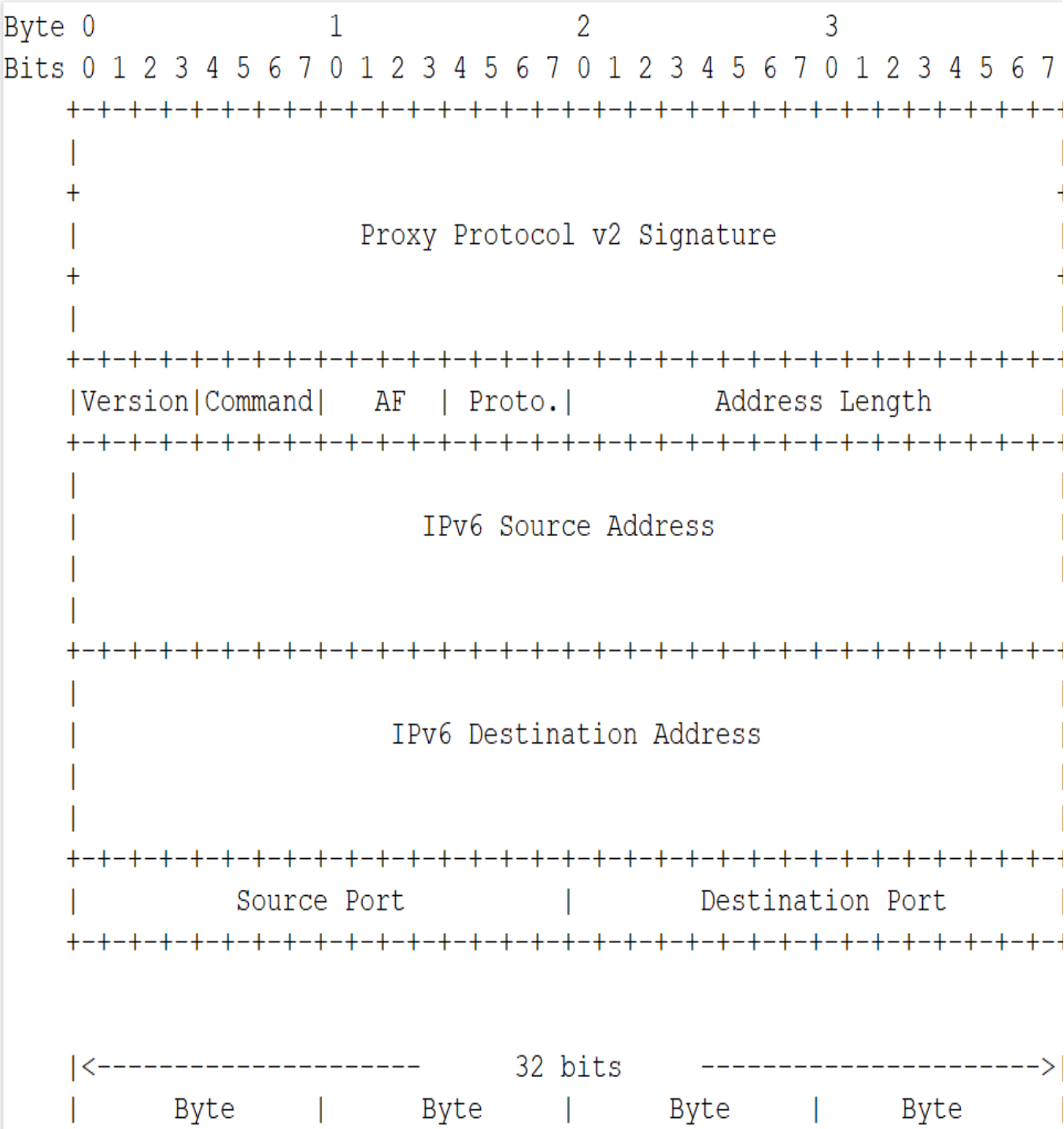
Proxy Protocol V2

Proxy Protocol V2 supports TCPv4, TCPv6, UDPv4 and UDPv6, and adopts the binary format. See details below:

IPv4



IPv6



Passing Real Client IP Through SPP

Last updated : 2024-12-25 16:26:37

Use Cases

The Simple Proxy Protocol (SPP for short below) is a custom protocol header format used for proxy servers to pass the real client IP and other related information to backend servers. It is applied in logging, access control, load balancing, troubleshooting, and other scenarios. The SPP header has a fixed length of 38 bytes, making it simpler compared to the Proxy Protocol V2.

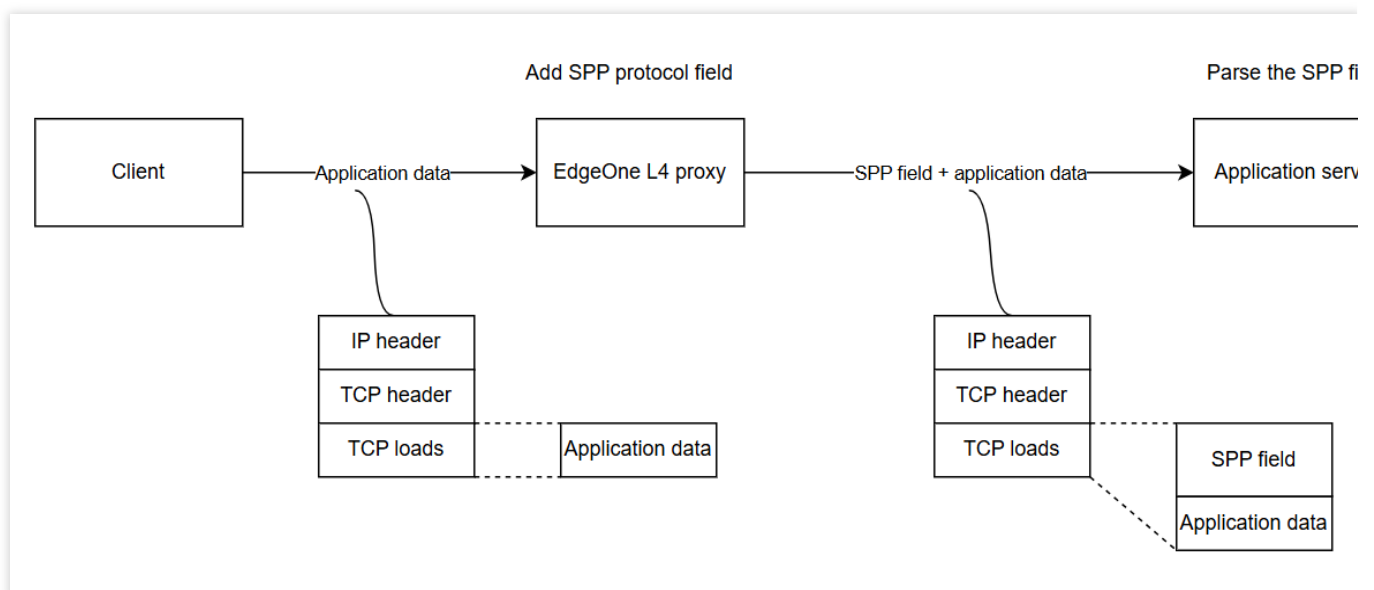
If your current backend service is a UDP service and already supports the SPP or you prefer a simpler parsing method, you can use the SPP to pass the real client IP. EdgeOne L4 proxy supports passing the real client IP to the business server based on the SPP standard. You can parse this protocol on the server side to obtain the real client IP and port.

Note:

The L4 proxy is only available with the Enterprise Edition package.

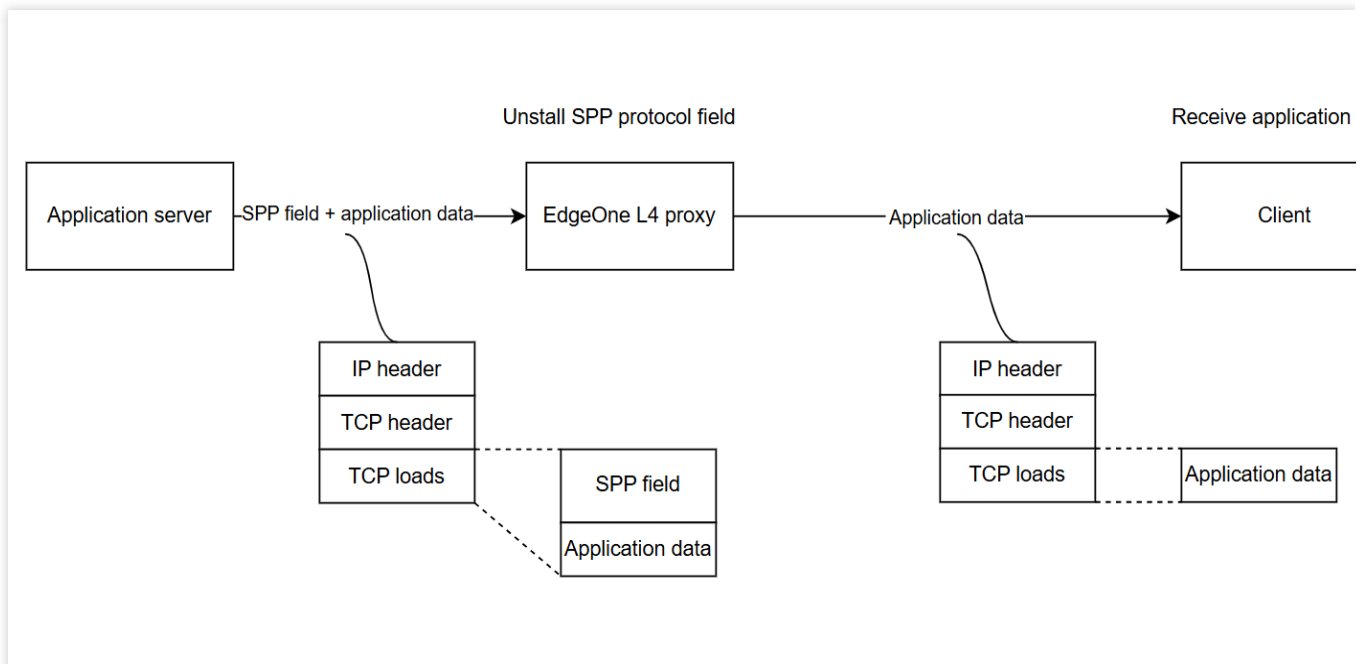
EdgeOne Handling Process for SPP

Requesting Access



As shown in the above figure, when you use the SPP to pass the client IP and port, the EdgeOne L4 proxy will automatically add the real client IP and port with a fixed length of 38 bytes before each payload according to the SPP header format. You can obtain the real client IP and port by parsing the SPP header field on the origin server.

Origin Server Response



As shown in the above figure, when the origin server responds, the response packet must carry the SPP header and be returned to the EdgeOne L4 proxy, which will automatically uninstall the SPP header.

Note:

If the origin server does not return the SPP header, it will cause the EdgeOne L4 proxy to truncate the business data in the payload.

Directions

Step 1: Configuring L4 Proxy Forwarding Rules

1. Log in into the [EdgeOne console](#), click **Site List** in the left sidebar, and then click the **site** you want to configure in the site list.
2. On the site details page, click **L4 Proxy**.
3. On the L4 proxy page, select the L4 proxy instance you want to modify and click **Configure**.
4. Select the Layer 4 proxy rule that requires passing the real client IP and click **Edit**.
5. Enter the corresponding business origin server address and port, select UDP for the forwarding protocol, select Simple Proxy Protocol for passing the client IP, and then click **Save**.

Forwarding rules

[Add rule](#)
[Batch import](#)
[Batch export](#)

| Rule ID | Forwarding port | Forwarding port | Origin type | Origin address | Origin port | Session persistence (seconds) | Pass client IP | Rule Tag | Status | Operation |
|-------------------|-----------------|-----------------|-------------|----------------|-------------|-------------------------------|----------------|----------|-----------|---|
| - | UDP | 6666 | Origin | 1.1.1.1 | 6666 | <input type="checkbox"/> | Simple Prox... | optional | - | Save |
| rule-2zsbo4ninu1d | UDP | 666 | Origin | 1.1.1.1 | 666 | Not enabled | Simple Prox... | - | Deploying | Edit Suspend Delete |

Do not pass

Step 2: Parsing the SPP Field on the Origin Server to Obtain the Real Client IP

You can refer to [SPP Header Format](#) and [Sample Code](#) for parsing the SPP field on the origin server. When the SPP is used to pass the real client IP, the format of the service packet data obtained by the server is as follows:

UDP payload (50 bytes)

Data (50 bytes)

Data: 56ec00000000000000000000ffff2ac1f6cb0000000000000000ffff2baf002a53a...

[Length: 50]

0000 00 00 00 01 00 06 fe ee 73 fa cc 34 00 00 08 00 s..4...

0010 45 b8 00 4e 5d b5 40 00 2f 11 a3 50 2b af 11 1c E..N].@./..P+...

0020 0a 00 03 17 82 ef 1a 0a 00 3a 85 ee 56 ec 00 00 :..V...

0030 00 00 00 00 00 00 00 00 ff ff 2a c1 f6 cb 00 00 *.....

0040 00 00 00 00 00 00 00 00 ff ff 2b af e0 02 a5 3a +.....:

0050 1a 0a 48 65 6c 6c 6f 20 53 65 72 76 65 72 ..Hello Server

SPP protocol header: 38 bytes in length

You can refer to the following sample code for parsing the business data to obtain the real client IP.

Go

C

```
package main

import (
    "encoding/binary"
    "fmt"
    "net"
)

type NetworkConnection struct {
    Magic      uint16
    ClientAddr net.IP
}
```

```
ProxyAddr net.IP
ClientPort uint16
ProxyPort uint16
}

func handleConn(conn *net.UDPConn) {
    buf := make([]byte, 1024) // Create a buffer.
    n, addr, err := conn.ReadFromUDP(buf) // Read the data packet from the conn

    if err != nil {
        fmt.Println("Error reading from UDP connection:", err)
        return
    }

    // Convert the received bytes to a NetworkConnection struct.
    nc := NetworkConnection{
        Magic:      binary.BigEndian.Uint16(buf[0:2]),
        ClientAddr: make(net.IP, net.IPv6len),
        ProxyAddr:  make(net.IP, net.IPv6len),
    }
    if nc.Magic == 0x56EC {
        copy(nc.ClientAddr, buf[2:18])
        copy(nc.ProxyAddr, buf[18:34])
        nc.ClientPort = binary.BigEndian.Uint16(buf[34:36])
        nc.ProxyPort = binary.BigEndian.Uint16(buf[36:38])

        // Print the SPP header information, including magic, real client IP
        fmt.Printf("Received packet:\n")
        fmt.Printf("\tmagic: %x\n", nc.Magic)
        fmt.Printf("\tclient address: %s\n", nc.ClientAddr.String())
        fmt.Printf("\tproxy address: %s\n", nc.ProxyAddr.String())
        fmt.Printf("\tclient port: %d\n", nc.ClientPort)
        fmt.Printf("\tproxy port: %d\n", nc.ProxyPort)
        // Print the actual and effective payload.
        fmt.Printf("\tdata: %v\n\tcount: %v\n", string(buf[38:n]), n)
    } else {
        // Print the actual and effective payload.
        fmt.Printf("\tdata: %v\n\tcount: %v\n", string(buf[0:n]), n)
    }

    // Respond with a packet. Note: The 38-byte SPP header must be returned complete
    response := make([]byte, n)
    copy(response, buf[0:n])
    _, err = conn.WriteToUDP(response, addr) // Send data.
    if err != nil {
        fmt.Println("Write to udp failed, err: ", err)
    }
}
```

```
}

func main() {
    localAddr, _ := net.ResolveUDPAddr("udp", ":6666") // Create a UDP address
    conn, err := net.ListenUDP("udp", localAddr)        // Create a listener.
    if err != nil {
        panic("Failed to listen for UDP connections:" + err.Error())
    }

    defer conn.Close() // Close the connection at the end.
    for {
        handleConn(conn) // Handle the incoming connection.
    }
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>
#define BUF_SIZE 1024
struct NetworkConnection {
    uint16_t magic;
    struct in6_addr clientAddr;
    struct in6_addr proxyAddr;
    uint16_t clientPort;
    uint16_t proxyPort;
};

void handleConn(int sockfd) {
    struct sockaddr_in clientAddr;
    socklen_t addrLen = sizeof(clientAddr);
    unsigned char buf[BUF_SIZE];
    ssize_t n = recvfrom(sockfd, buf, BUF_SIZE, 0, (struct sockaddr *)&clientAddr,
    if (n < 0) {
        perror("Error reading from UDP connection");
        return;
    }
    // Convert the received bytes to a NetworkConnection struct.
    struct NetworkConnection nc;
    nc.magic = ntohs(*(uint16_t *)buf);
    if (nc.magic == 0x56EC) { // Magic with the value 0x56EC indicates an SPP header
        memcpy(&nc.clientAddr, buf + 2, 16);
        memcpy(&nc.proxyAddr, buf + 18, 16);
        nc.clientPort = ntohs(*(uint16_t *) (buf + 34));
    }
}
```

```

    nc.proxyPort = ntohs(*(uint16_t *) (buf + 36));
    printf("Received packet:\n");
    printf("\t\tmagic: %x\n", nc.magic);
    char clientIp[INET6_ADDRSTRLEN];
    char proxyIp[INET6_ADDRSTRLEN];
    inet_ntop(AF_INET6, &nc.clientAddr, clientIp, INET6_ADDRSTRLEN);
    inet_ntop(AF_INET6, &nc.proxyAddr, proxyIp, INET6_ADDRSTRLEN);

    // Print the SPP header information, including magic, real client IP and po
    printf("\t\tclient address: %s\n", clientIp);
    printf("\t\tproxy address: %s\n", proxyIp);
    printf("\t\tclient port: %d\n", nc.clientPort);
    printf("\t\tproxy port: %d\n", nc.proxyPort);
    // Print the actual and effective payload.
    printf("\t\tdata: %.*s\n\t\tcount: %zd\n", (int)(n - 38), buf + 38, n);
} else {
    printf("\t\tdata: %.*s\n\t\tcount: %zd\n", (int)n, buf, n);
}
// Respond with a packet. Note: The 38-byte SPP header must be returned complet
sendto(sockfd, buf, n, 0, (struct sockaddr *)&clientAddr, addrLen);
}

int main() {
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Failed to create socket");
        exit(EXIT_FAILURE);
    }
    // Create a UDP address using the local address and port.
    struct sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(6666);
    if (bind(sockfd, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0) {
        perror("Failed to bind");
        exit(EXIT_FAILURE);
    }
    while (1) {
        handleConn(sockfd);
    }
}

```

Step 3: Testing and Verification

You can use a server as the client, construct client requests, and use the nc command to simulate UDP requests. The details of the command are as follows:

```
echo "Hello Server" | nc -w 1 -u <IP/DOMAIN> <PORT>
```

Here, IP/Domain indicates the access IP or domain name of your L4 proxy instance. You can view the corresponding information of the L4 proxy instance in the EdgeOne console. Port indicates the forwarding port configured for the rule in [Step 1](#).

Create L4 proxy instance

| sid-2zsb03x5qeoh | Scheduling mode | Proxy mode | Forwarding rules | Status | Service area | Update time | Operation |
|------------------|-----------------|------------------------------------|------------------|-----------|-----------------------|---------------------|---|
| sid-2zsb03x5qeoh | CNAME | DDoS Protection L4 acceleration | 1 | Deploying | Global (MLC excluded) | 2024-07-19 16:10:11 | View Disable Delete |

Total items: 1

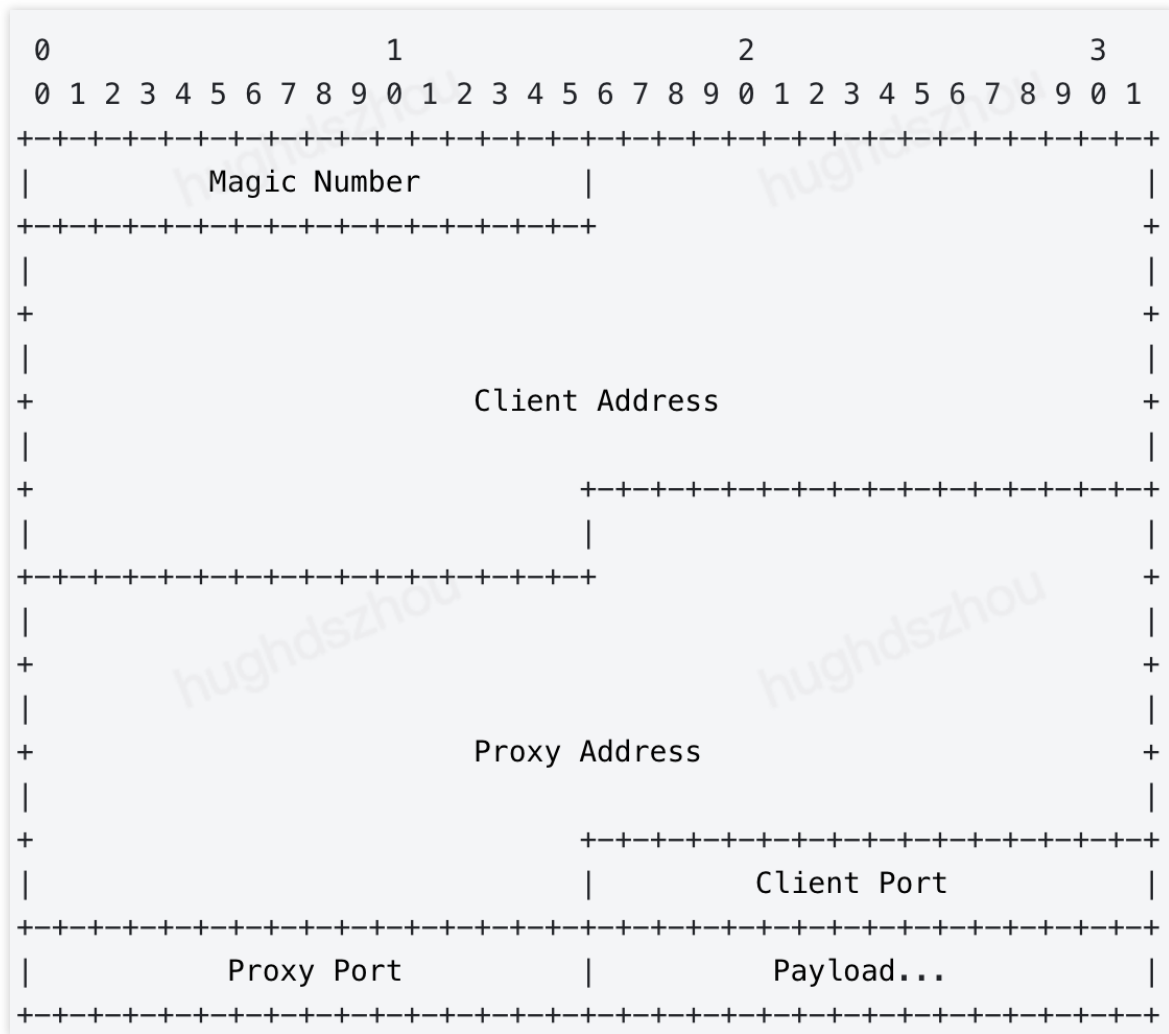
10 / page 1 / 1 page

The server receives the request and parses the client IP address as follows:

```
[root@VM-3-23-centos services]# ./server
Received packet:
  magic: 56ec
  client address: 42.193.246.203
  proxy address: 43.175.224.2
  client port: 34394
  proxy port: 6666
  data: Hello Server
  count: 50
```

Related References

SPP Header Format



Magic Number

In the SPP format, Magic Number is 16 bits long with a fixed value of 0x56EC. It is mainly used to identify the SPP and also specify the fixed length of SPP header to be 38 bytes.

Client Address

IP address of the client initiating a request, which is 128 bits long. If the request is initiated by an IPV4 client, the value indicates IPV4; if initiated by an IPV6 client, the value indicates IPV6.

Proxy Address

IP address of the proxy server, which is 128 bits long and can be parsed in the same way as the Client Address.

Client Port

Port for the client to send UDP packets, which is 16 bits long.

Proxy Port

Port for the proxy server to receive UDP packets, which is 16 bits long.

Payload

Actual data following the header in a packet.