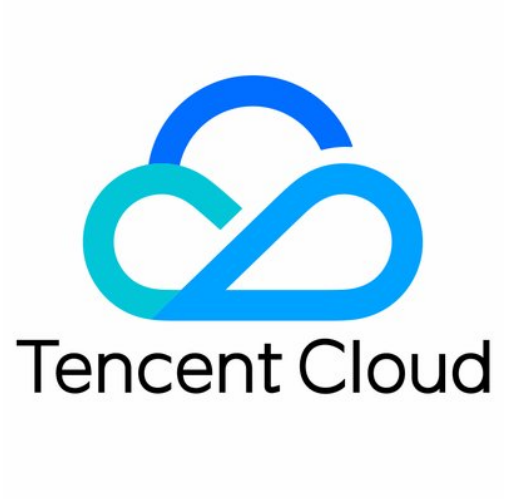
Cloud Load Balancer

More

Product Documentation



Contents

More
HTTP Long Connections

Last updated : 2020-03-10 10:40:57

Currently, HTTP persistent connection for Layer-7 CLB can be configured to the default value of 75s. You can customize the configuration for different CLB instances. The following describes HTTP persistent connection and non-persistent connection.

Relationship between HTTP protocol and TCP/IP protocol

HTTP persistent and non-persistent connections are essentially TCP persistent and non-persistent connections. HTTP is an application layer protocol. TCP is used at the transport layer, and IP is used at the network layer. IP protocol mainly solves network routing and addressing problems. TCP protocol mainly solves how to reliably transfer data packets at the IP layer, so all packets can be received in the order they are sent. TCP protocol is reliable and connection-oriented.

What does stateless HTTP protocol mean?

HTTP protocol is stateless, meaning the protocol has no memory for transaction processing, and the server does not know the state of the client. In other words, opening a webpage on a server has nothing to do with the previous opening of a webpage on the same server. HTTP is a stateless and connection-oriented protocol. It does not mean HTTP cannot maintain TCP connections, or it uses the UDP protocol (no connection).

What are persistent and non-persistent connections?

Non-persistent connection is used by default in HTTP/1.0. A connection is established each time the client and server perform an HTTP operation, while terminated when the task ends. If a client browser accesses a webpage of HTML or other formats that contains web resources (such as JavaScript, image, or CSS files), the browser will create an HTTP session each time it encounters such a web resource.

Starting from HTTP/1.1, persistent connection is used by default to maintain the connection. The following code is added in the response header when using HTTP persistent connection:

```
Connection:keep-alive
```

When persistent connection is used, TCP connection used to transfer HTTP data between the client and the server will not be closed after a webpage is opened. When the client accesses the server again, the established connection will be used. `Keep-Alive` does not keep the connection permanently alive, but the time can be configured in different server softwares (such as Apache). To implement persistent connection, both the client and the server should support persistent connection.

HTTP persistent connection and non-persistent connection are essentially TCP persistent connection and non-persistent connection.

TCP Connection

If TCP protocol is used for network communication, a connection must be established between the client and the server before an actual read/write operation is performed. After the operation is completed, the connection can be released if no longer needed by the two parties. The establishment of a connection relies on a "three-way handshake", while the release requires a "four-way handshake". The establishment of each connection consumes resource and time.

Diagram of connection establishment via three-way handshake:

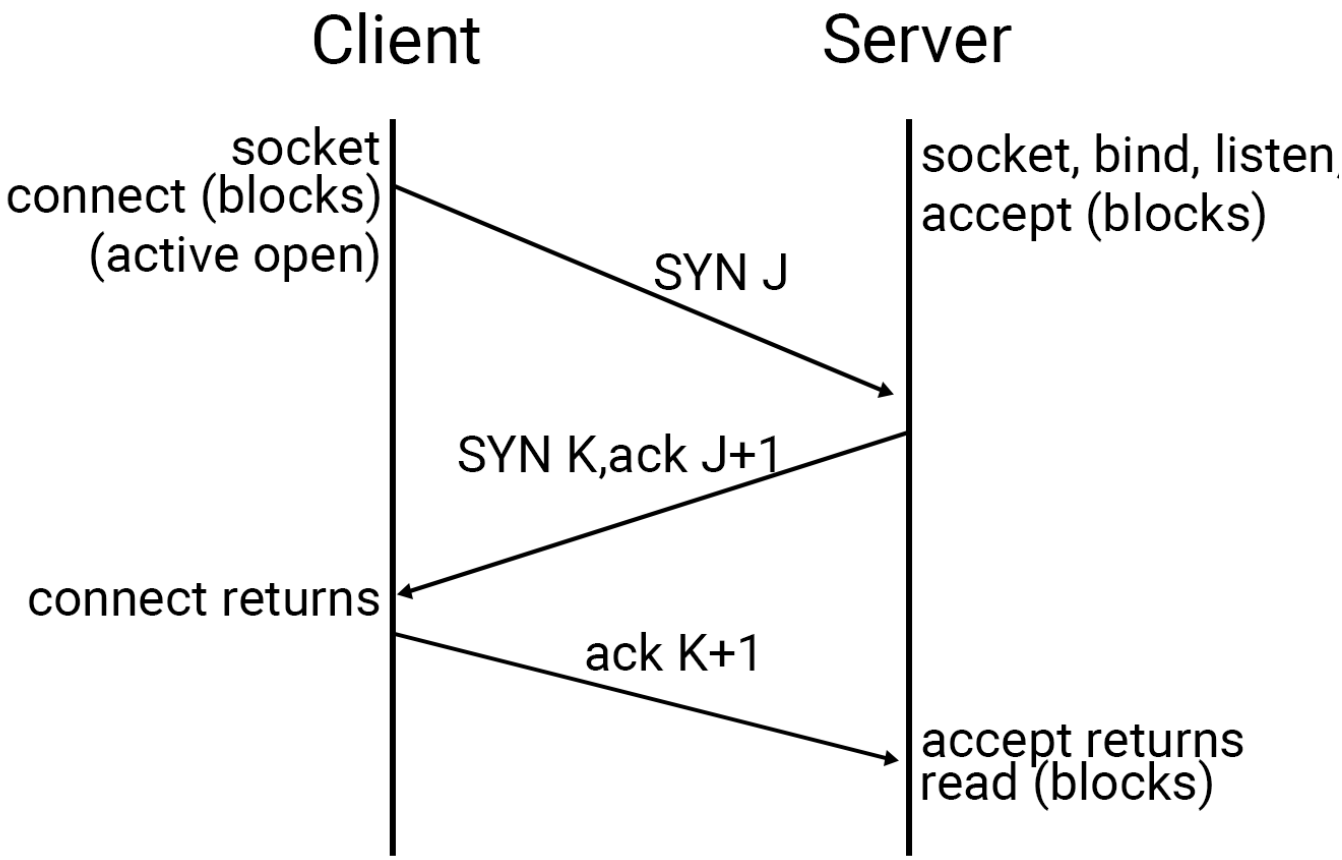
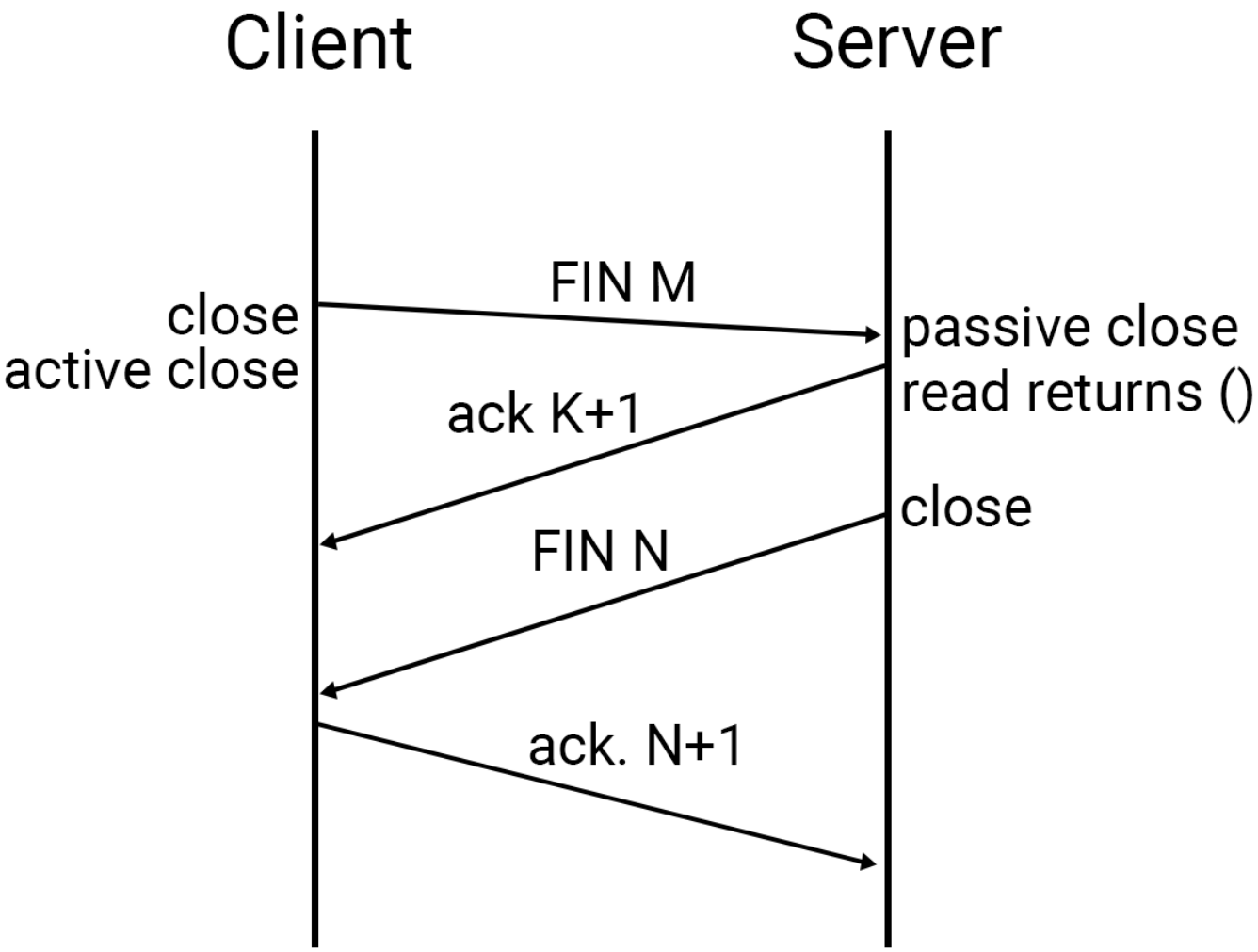


Diagram of connection close via four-way handshake:



TCP non-persistent connection

Below is a simulation of TCP non-persistent connection. The client initiates a connection request to the server, the server receives the request, and a connection is established. The client sends a message to the server, the server responds to the client, and a request is completed. Either party can now initiate a close operation, which is usually initiated by the client first. In general, non-persistent connection only passes one single request between the client and the server.

Non-persistent connections are easy to manage, and all existing ones are useful connections that do not require additional control.

TCP persistent connection

Below is a simulation of persistent connection. The client initiates a connection request to the server, the server accepts the request, and a connection is established. After a request between the client and the server is completed, the connection will not be closed but used for subsequent read and write operations.

TCP keep-alive feature is mainly provided for server applications. If the client has disappeared but the connection is not closed, a half-open connection will be retained on the server, which will wait for data from the client. In this case, the server will keep waiting for data from the client. The keep-alive feature detects half-open connections on the server.

If a given connection has not taken any actions within two hours, the server will send a probe message segment to the client to detect the following four client statuses based on responses from the server host:

- The client host is still running and the server is reachable. TCP response from the client is normal, and the server will reset the keep-alive timer.
- The client host has crashed and shut down or is restarting. The client cannot respond to TCP, and the server will not receive the client's response to the probe. The server will send a total of 10 probes at an interval of 75 seconds. If the server does not receive any response, it assumes the client has closed and terminated the connection.
- The client has crashed and restarted. The server will receive a response to its keep-alive probe. This response is a reset for the server to terminate the connection.
- The client is running normally but the server is unreachable. This status is similar to the second one.

Pros and cons of persistent and non-persistent connection

Persistent connection can reduce the number of TCP connection establishment and closing, saving time and resources. Persistent connection is suitable for clients that frequently request resources. When persistent connection is used, the client generally does not close the connection. As the number of client connections increases, however, the server will maintain too many connections. In this case, the server may need to close connections that have no requests for a long time, in order to prevent malicious connections from damaging the server. If possible, you can limit the maximum number of persistent connections for each client to avoid malicious clients from damaging the real server.

Non-persistent connections are relatively easy for the server to manage. All existing connections are useful and do not require additional control. However, if client requests are frequent, time and bandwidth will be wasted on the establishment and closing of TCP connections.

Persistent and non-persistent connections are generated based on closing policies adopted by the client and server. The optimal policy varies by use case.

WebSocket Principle

Last updated : 2020-03-09 14:59:41

Communication of a web application is as follows: the client sends a request via the browser, the server receives the request, processes it, and returns the result to the client. The client's browser then displays the information. This mechanism better support applications with infrequent information change. However, it may be less effective for applications with high requirement for real-time transmission and high concurrence. Given the robust growth of mobile internet, web application frequently encounters high concurrence and demands for real-time response. This includes real-time information on financial securities, geographic information in web-based navigation apps, and real-time message push in social network.

To manage these business scenarios, web development based on request-response mode generally uses real-time communication method such as polling. When using polling, the client frequently sends requests to the server at certain intervals to keep data synced between the client and server. But when the client sends requests to the server at fixed intervals, data on the server may be outdated, generating a large number of unnecessary requests that waste bandwidth and reduce efficiency.

Adobe Flash uses its embedded Socket to implement data exchange and Flash to expose related APIs for JavaScript to call, achieving real-time communication. This method is more efficient than polling, because Flash is commonly installed and widely used. However, Flash is not well supported on mobile devices. iOS does not support Flash, while Android provides poor support for Flash with high requirements for hardware configuration. In 2012, Adobe announced that Flash no longer supported Android 4.1 and above, marking the end of Flash on mobile devices.

Traditional web modes would encounter difficulties when handling demands with high concurrence and real-time communication. A bi-directional communication mechanism with cost efficiency was required to facilitate real-time data transmission. WebSocket, known as "web TCP", was thereby developed based on HTML5. At the early stage, the industry had no unified HTML5 specification. Browser and application server vendors had different implementation methods, such as MQTT of IBM and Comet’s open-source framework. In 2014, HTML5 was established as a standard specification. Application server and browser vendors gradually unified their HTML5 implementation, and WebSocket protocol was implemented in Java EE 7. At this point, WebSocket for clients and servers is fully established. For more information on the new HTML protocol and WebSocket support, see HTML5 Specification.

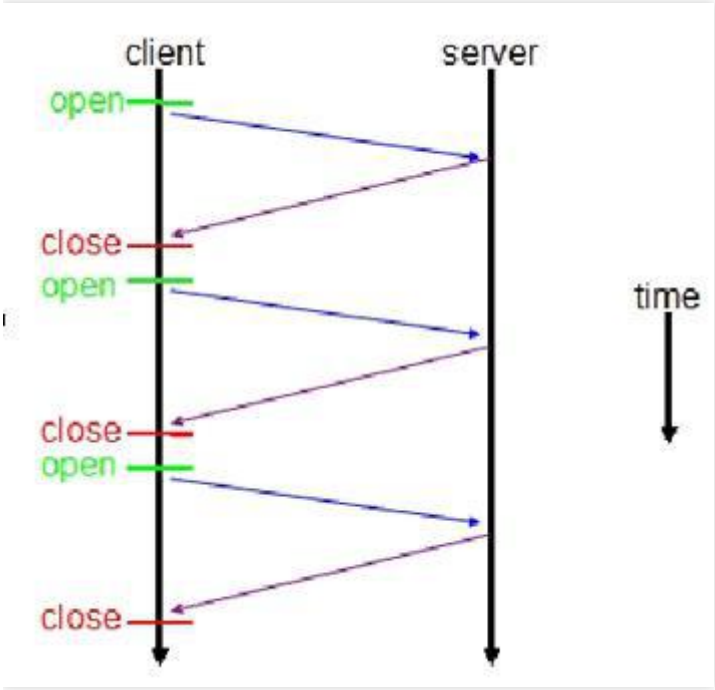
WebSocket Mechanism

The following is an overview of the principle and operating mechanism of WebSocket.

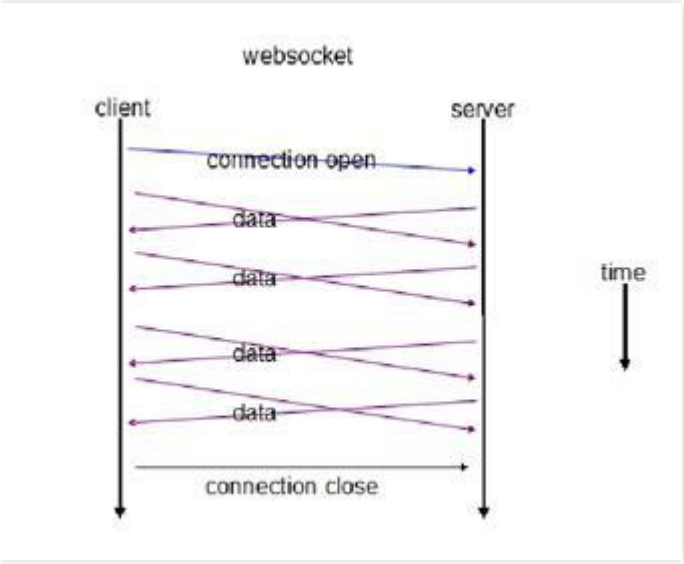
WebSocket is a new HTML5 protocol, which implements full-duplex communication between browser and server. It supports real-time communication and reduces the consumption of server resources as well as bandwidth. Similar to HTTP, WebSocket uses an established TCP connection to transmit data. However, their biggest differences are:

- WebSocket is a bi-directional communication protocol. After a connection is established, WebSocket server and client can actively send or receive data to and from each other, which is similar to Socket.
- Like TCP, a connection should be established first with WebSocket before communication can be made.

The traditional HTTP request-response mode between server and client is as shown below:



The WebSocket client-server request-response mode is as shown below:



In traditional HTTP mode, each request-response requires a new connection between the client and server. WebSocket, which is similar to Socket, is a communication mode based on persistent TCP connection. Once a WebSocket connection is established, subsequent data will be transmitted in the form of frame sequence. The client or server does not need to initiate a connection request again before WebSocket is disconnected. In scenarios with high concurrence or high volumes of load traffic on client and server, WebSocket significantly reduces consumption of network bandwidth resources and improves performance. In addition, the client sends and receives messages over a persistent connection, achieving real-time communication.

Compared to HTTP persistent connection, WebSocket has the following advanatges:

- It is a real full-duplex method. The client and server are completely equal after a connection is established. They can initiate requests proactively. In contrast, HTTP persistent connection is based on HTTP. It uses the traditional mode where the client initiates a request to the server.
- In addition to real data, a large number of HTTP headers are exchanged between the client and server in HTTP persistent connection, which reduces data exchange efficiency. In WebSocket protocol, data can be exchanged without HTTP headers after a TCP connection is established through the first request. Given this difference, WebSocket can be implemented after client and server upgrades (mainstream browsers already support HTML5). WebSocket also has distinct features such as multiplexing and reusable WebSocket connection by different URLs.

Below is a comparison between WebSocket and traditional HTTP based on messages exchanged between the client and the server:

The new WebSocket instantiates a new WebSocket client object on the client and requests a WebSocket URL similar to ws://yourdomain:port/path from the server. The client WebSocket object automatically parses and recognizes the WebSocket request, connects the server port, and implements handshake. The client sends data in the following format:

```
GET /webfin/websocket/ HTTP/1.1
Host: localhost
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: xqBt3ImNzJbYqRINxEFlkg==
Origin: http://localhost:8080
Sec-WebSocket-Version: 13
```

As can be seen above, the WebSocket connection message initiated by the client is similar to the traditional HTTP message. `Upgrade: websocket` indicates a WebSocket request, and `Sec-WebSocket-Key` indicates a Base64-encoded ciphertext sent by the WebSocket client. The server must return a corresponding encrypted `Sec-WebSocket-Accept` response. Otherwise, the client will throw an `Error during WebSocket handshake` error and close the connection.

After receiving the message, the server returns data in the following format:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: K7DJLdLooIwIG/MOpvWFB3y3FE8=
```

The value of `Sec-WebSocket-Accept` is calculated by the server using the same key as that of the client and then returned to the client. `HTTP/1.1 101 Switching Protocols` indicates that the server accepts the connection from the client over WebSocket. After this request-response, WebSocket handshake between the client and server is completed. TCP communication can now be implemented. For more information on data exchange format between WebSocket client and server, see WebSocket Protocol Stack.

WebSocket API is easy to use in development. You only need to instantiate WebSocket and create a connection. The server and client can then send messages to each other and respond accordingly. For more information on WebSocket API and code implementation, see WebSocket implementation and case analysis sections.

Tencent Cloud offers Layer-7 public network CLB for daily rental whose forwarding module supports WebSocket. Multiple customers such as Heroes Evolved and Yinhan Games have connected to this service. If incompatibility occurs, modify the WebSocket configuration. The WebSocket server does not verify circled fields in the figure below:

```
GET / HTTP/1.0
Upgrade: websocket
Connection: upgrade
Host: 21319
Pragma: no-cache
Cache-Control: no-cache
Origin: chrome-extension://pfdhoblngboilpfeibdedpjgfnlcodoo
Sec-WebSocket-Version: 13
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-CN,zh;q=0.8
Sec-WebSocket-Key: qhiuFIg2USbwzjyKs28zrg==
Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: TgxnYhlXJHsvWZbPJEleQA8+XNo=
```

To use WebSocket in your video business, follow the steps below:

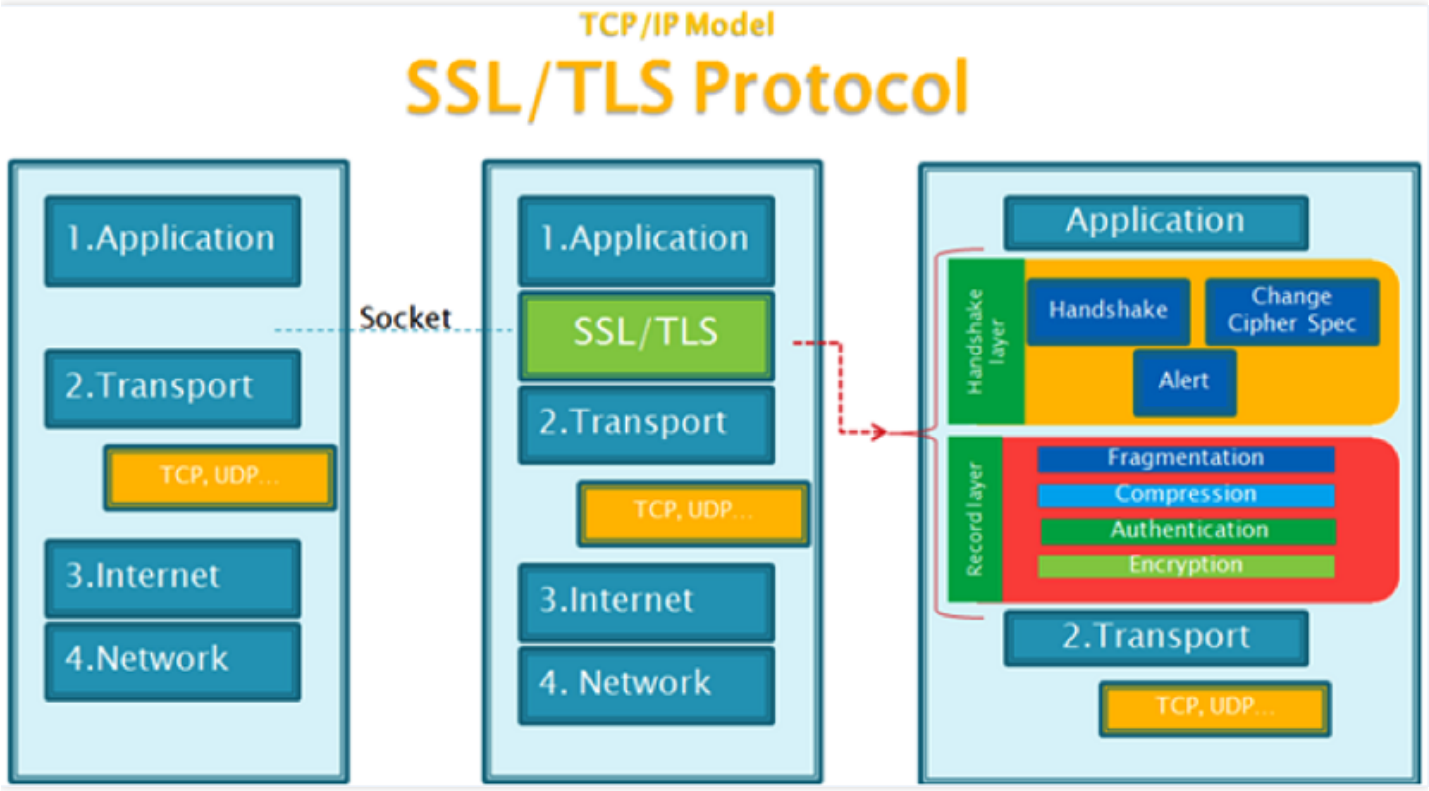
- Use heartbeat to maintain WebSocket linkage and detect whether influencers or VJs on the client are online.
- Configure `proxy_read_timeout` of Layer-7 CLB instance to 60s.
- Configure the heartbeat to 50s, so WebSocket can be connected for the long term.

Custom configuration for WebSocket is coming soon.

SSL Principle

Last updated : 2020-03-09 14:59:41

SSL/TLS is an optional protocol between the application layer (HTTP) and the transport layer (TCP). Its protocol architecture is as shown below:



If HTTP communication does not use SSL/TLS, all information will be transmitted in plaintext, which involves the following risks:

- Eavesdropping: A third party can access to the communication content.
- Tampering: A third party can modify the communication content.
- Pretending: A third party can pretend to be someone else to participate in the communication process.

SSL/TLS protocol is designed to address these risks with the following goals:

- All information is encrypted and cannot be eavesdropped by a third party.
- A verification mechanism helps both parties in the communication immediately detect tampering.
- Identity certificates will be used to authenticate the identity.

Currently, all mainstream browsers support SSL/TLS.

Operating process of SSL/TLS protocol

SSL/TLS protocol uses public-key encryption. The client requests the public key from the server first, and uses it to encrypt as well as send the information to the server. After receiving the ciphertext, the server decrypts it with its private key.

There are two problems to be solved:

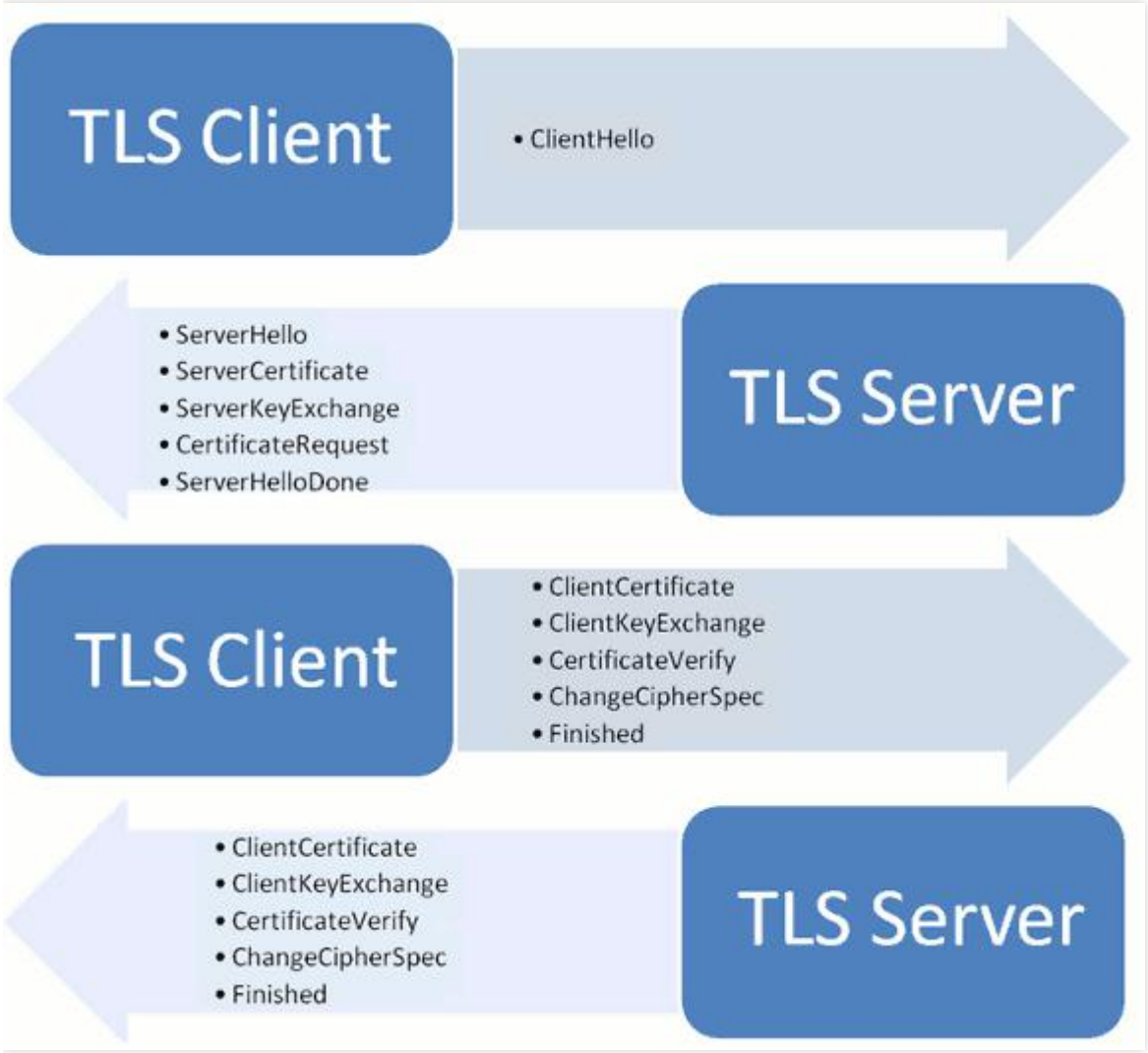
- How to ensure that the public key is not tampered with?
  Solution: Put the public key in a digital certificate. As long as the certificate is trustworthy, the public key will be trustworthy.
- How to reduce the time consumed as public-key encryption involves high volumes of calculation?
  Solution: For each session, the client and server will generate a "session key" and use it to encrypt the information. As the "session key" is symmetrically encrypted, the calculation is fast. The server's public key is only used to encrypt the "session key", reducing encryption time.

SSL/TLS protocol works as follows:

- The client requests and verifies the public key from the server.
- The two parties negotiate to generate a "session key".
- The two parties use the "session key" for encrypted communication.

The first two steps are known as "handshake".

Specific handshake process



A "handshake" involves four stages of communication where all information is transmitted in plaintext.

Client Request (ClientHello)

The client (usually a browser) first sends a request for encrypted communication to the server, which is generally called a “ClientHello” request.

In this step, the client mainly provides the following information to the server:

- Supported protocol versions, such as TLS 1.0.
- A client-generated random number that will be used to generate a "session key" later.
- Supported encryption methods, such as RSA public key encryption.
- Supported compression methods.

The information sent by the client does not contain the domain name of the server. In other words, a server can only host one website in principle. Otherwise, it would be difficult to tell which website’s

digital certificate should be provided to the client. This is why each server can only have one digital certificate.

This is inconvenient for users of virtual host. To solve this, Server Name Indication was added in 2006 as an extension to TLS protocol, allowing the client to provide the server with the requested domain name.

Server Response (ServerHello)

After the server receives the client request, it sends a response to the client, which is generally called “ServerHello”. The server's response contains the following information:

- The version of the encrypted communication protocol to be used, such as TLS 1.0. If the browser and server support different versions, the server will shut down encrypted communication.
- A server-generated random number that will be used to generate a "session key" later.
- The encryption method to be used, such as RSA public key encryption.
- Server certificate.

If the server needs to confirm the client identity, it will also request a "client certificate". For example, financial institutions usually only allow authenticated customers to connect to their networks. Their customers are provided with USB keys, which contain client certificates.

Client Response

After receiving the server response, the client will verify the server certificate. If the certificate is not issued by a trustworthy CA, the domain name in the certificate is not the same as the actual domain name, or the certificate has expired, a warning will be shown to the requester, who can choose whether to continue the communication.
If the certificate has no issues, the client will get the server's public key from the certificate and send the following information to the server:

- A random number, which is encrypted with the server's public key to prevent eavesdropping.
- Encoding change notification, indicating subsequent messages will be sent using the encryption method and key agreed upon by both parties.
- Client handshake end notification, indicating handshake on the client side has ended. This is also the hash value of all the content sent previously and used by the server for verification.

The random number mentioned above is the third random number used during the entire handshake process, which is also known as a "pre-master key". Both the client and server now have three random numbers. They can generate the same "session key" used for this session via the encryption method agreed upon in advance.

For RSA key exchange algorithms, the pre-master key is a random number. When it is coupled with the two random numbers in “hello” messages, a final symmetric key will be generated through key derivation.

The pre-master key exists because SSL protocol does not trust that every host can generate a completely random number. The key generated by the random numbers of the client, the server, and the pre-master key will be difficult to guess. One pseudo-random number may not be completely random, but three together will create a random number.

In addition, if the server requests a client certificate in the previous step, the client will send the certificate and related information in this step.

Final response from the server

After receiving the pre-master key (the third random number) from the client, the server calculates and generates the "session key" used for this session. It then sends the following information to the client:

- Encoding change notification, indicating subsequent messages will be sent using the encryption method and key agreed upon by both parties.
- Server handshake end notification, indicating handshake on the server side has ended. This is also the hash value of all the content sent previously and used by the client for verification.

The entire handshake process has now ended. The client and the server begin encrypted communication. HTTP protocol is used, except that the content is encrypted with the "session key".

Below is a simple example to illustrate the process. Assume that SSL client A communicates with SSL server B. The encrypted message is enclosed in square brackets "[]" to distinguish from plaintext message. The description of actions by both parties is enclosed in parentheses "()".

- A:

```
”I want to talk to you securely. My symmetric encryption algorithms are DES and RC5, my key exchange algorithms are RSA and DH, and my message digest algorithms are MD5 and SHA.”
```

- B:

```
”Let’s use the DES-RSA-SHA combination. This is my certificate, which contains my name and public key. Please use it to verify my identity.”
```

Send the certificate to A.

```
”There is nothing else to say for now.”
```

- A:

Check whether the name of B in the certificate is correct, and verify the authenticity of B's certificate with the existing CA certificate. If something is wrong, it will issue a warning and close the connection. This step guarantees that B's public key is authentic.

Generate a secret message, which will be processed and then used as the encryption key to encrypt the initialization vector (IV) and the key for HMAC. This secret message (called `per_master_secret` in the protocol) is encrypted with B's public key and encapsulated into a message called `ClientKeyExchange` . Using B's public key prevents eavesdropping.

```
"I generated a secret message and encrypted it with your public key. Here it is." Send `ClientKeyExchange` to B. "Please note that I will send you a message in an encrypted manner!"
```

Process the secret message to generate the encryption key, and encrypt the IV and the key for HMAC.

```
[Over.]
```

- B:

Use its own private key to decrypt the secret message in `ClientKeyExchange` , process it to generate the encryption key, and encrypt the IV and the key for HMAC. The two parties have now securely agreed on an encryption method.

```
"Please note that I'm going to send you a message in an encrypted manner!"
```

```
[Over]
```

- A:

```
[My secret is...]
```

- B:

```
[No one else can eavesdrop that...]
```

Session Persistence Principle

Last updated : 2020-03-10 10:42:01

What is session persistence?

Session persistence is a common and rather complex topic in CLB. Sometimes, session persistence is referred to as sticky session. It is a mechanism that can identify client-server affinity and guarantee that associated access requests will be allocated to the same server during load balancing.

When is session persistence needed?

First, we need to understand some concepts: connection, session and their differences. Note that connection and session mean the same if we only talk about load balancing.

To put it simple, it is a session if users need login. Otherwise, it is a connection.

For data packets in the same connection, CLB will perform network address translation (NAT) on them and forward them to a fixed real server for further processing. A table in CLB system records information about the connection status, such as **source IP: port**, **target IP: port**, **server IP: port**, and idle timeout period (`Idle Timeout`). Because the table used to record connection status consumes system memory resources, its size cannot be infinite. All traditional vendors will have certain limits on its size, which is called the maximum number of concurrent connections, i.e., the maximum number of connections that the system can sustain at the same time. In CLB's current connection status table, an `Idle Timeout` parameter is provided. When a connection has no traffic within the idle timeout period, CLB will automatically delete it and release system resources.

After the connection is deleted, client requests may not be sent to the same real server. Instead, the traffic distribution policy of CLB prevails.

In some scenarios where login is necessary, a session is required between the client and server to record client information.
For example, in most ecommerce application systems or online systems that require identity verification, a transaction or request can be completed after several rounds of interactions between the client and server. As these interactions are closely related, the server needs to know the results of the last one or several interactions before performing a new one. This requires that all these interactions are completed on the same server, instead of being distributed to different servers by CLB. Otherwise, the following exceptions may occur:

- The client enters the correct username and password but is repeatedly redirected to the login page.
- The client enters the correct verification code but repeatedly receives a verification code error notification.
- Items added to the shopping cart in the client are lost.

Session persistence is required to ensure that requests from the same client will be forwarded to the same real server for processing in certain situations. In other words, multiple connections established between the client and servers are sent to the same server for processing. If load balancer is deployed between the client and servers, these connections will likely be forwarded to different servers for processing. If there is no session information synchronization mechanism between servers, other servers cannot authenticate the user and exceptions may occur when the user interacts with the application system.

CLB forwards requests and connections from clients evenly to multiple servers to avoid overload on a single server. The session persistence mechanism requires certain requests to be forwarded to the same server for processing. In actual deployment environment, an appropriate session persistence mechanism needs to be selected based on the application environment.

Session persistence types

Session persistence based on source address (Layer-4 session persistence)

Session persistence based on source address (also called IP-based session persistence or simple session persistence) is when CLB uses the source address of the access request as the basis to evaluate the associated session. All access requests from the same IP address will be forwarded to the same server during load balancing.

An important parameter in simple session persistence is connection timeout period, which is a time value configured by CLB for each connected session. If the time interval between the last completion of the session and the next is less than this value, CLB will persist the session for the new connection. Otherwise, CLB will consider the new connection as a new session and perform load balancing.

Simple session persistence can be easily implemented based on Layer-3 and Layer-4 information of data packets, improving efficiency.

NginX can support simple session persistence
ip_hash
Requests are distributed according to hash results of accesing IPs, ensuring that each visitor can regularly visit one RS so as to solve the problem of session.
For example:

```
upstream bakend {
ip_hash;
server 192.168.0.14:88;
server 192.168.0.15:80;
}
```

However, the problem with this method is when multiple clients access servers through proxy or address translation, requests from the same source address will be allocated to the same server, resulting in severe load imbalance between servers.

In another situation, a client generates a large number of concurrent requests that need to be allocated to multiple servers for processing while persisting the session. In this case, session persistence based on source client address will cause load balancing to fail.

If the above problems occur, you must consider using other types of session persistence.

Cookie-based session persistence (Layer-7 session persistence)

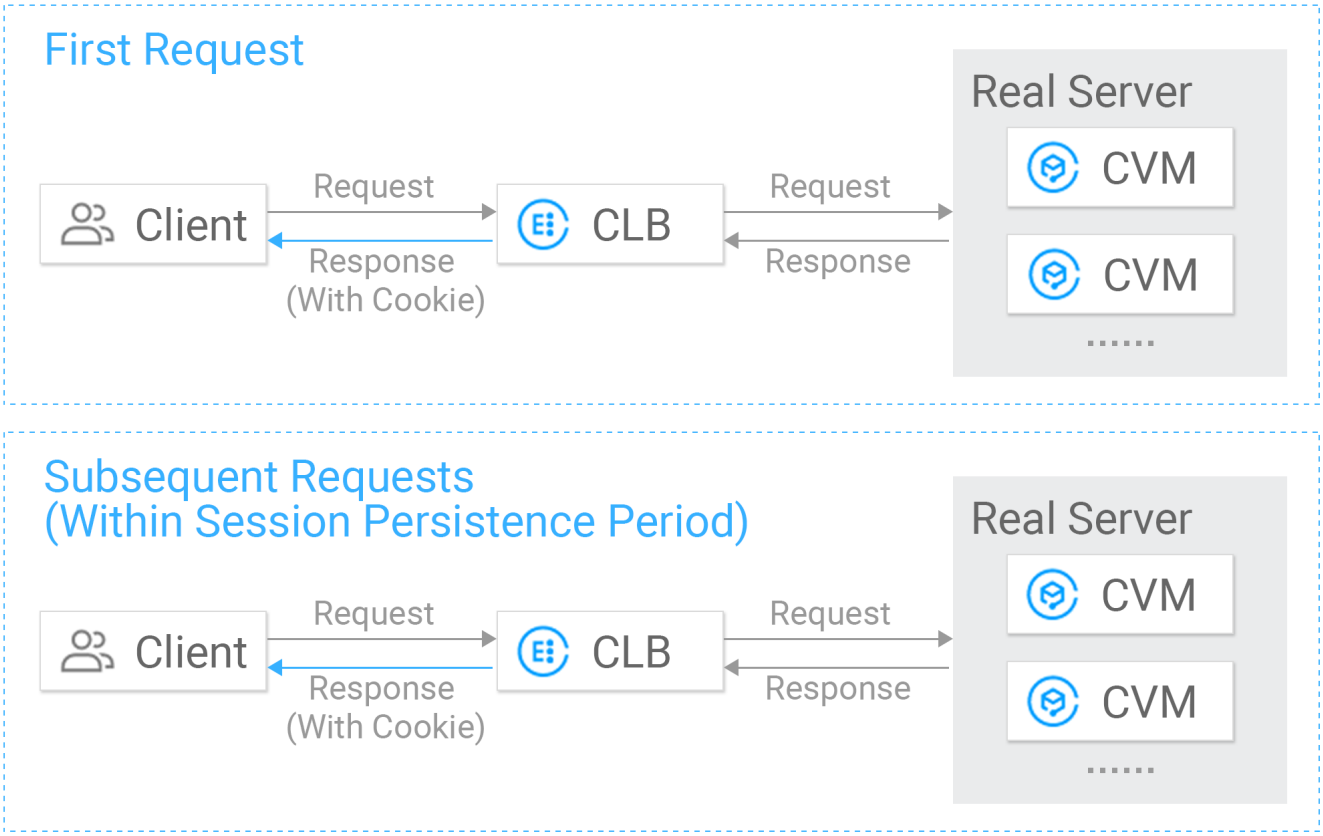
In cookie-based session persistence, the client and load balancer process the cookie, which is transparent to the real server.

Processing logic for initial client request

1. When the client initiates a request for the first time, the request header does not have the session persistence cookie. CLB receives the request and finds that the request header has no session persistence cookie, so it will select a server based on the polling algorithm.
2. The real server processes the request and returns the response header and content to the CLB instance.
3. CLB generates a cookie and stores it in the memory.
4. CLB returns the request response to the client.

Processing logic for subsequent client requests during the session persistence period

1. When the client sends a new request, the request must carry a session persistence cookie.
2. CLB receives the cookie, finds through the algorithm the real server that processes the previous request, and forwards the request to that server.



Session Persistence

This method implements session persistence simultaneously with load balancing by sharing a session among multiple real servers.

1. Stored in database
   The session information can be stored in a database table for sharing among application servers. This method is suitable for websites with low database access requests.
   - Advantage: easy to implement.
   - Disadvantage: compared to an application server, database server is harder to scale out and its resources are more valuable. In web applications with high concurrence, the performance of database server is usually affected. If sessions are stored in database tables, frequent database operations will affect the business.
2. Stored in file system
   A file system such as NFS can be used to share sessions among multiple servers. This method is suitable for websites with low concurrence.
   - Advantage: only disks that store sessions need to be mounted to servers, which is easy to implement.

- Disadvantage: NFS cannot provide high performance for high-concurrence reads/writes. Its disadvantages are in hard disk I/O performance and network bandwidth, especially for frequent reads/writes on small files such as sessions.

3. Stored in Memcached

   Memcached can be used to store session data, which is read directly from the memory.

   - Advantage: high efficiency and faster session reads/writes compared to when stored in file system. Sharing sessions among servers is also more convenient, as servers can be simply configured to use the same group of Memcached servers to reduce additional workload.
   - Disadvantage: once Memcached is down, data in the memory will be lost, even though it is not a serious issue for session data. If the number of access requests to a website is high and there are too many sessions, Memcached will delete those that are not commonly used. If the user resumes use after a certain period of time, however, the session will fail to be read.

How Cookies Work

Last updated : 2020-04-01 17:15:14

Cookie Overview

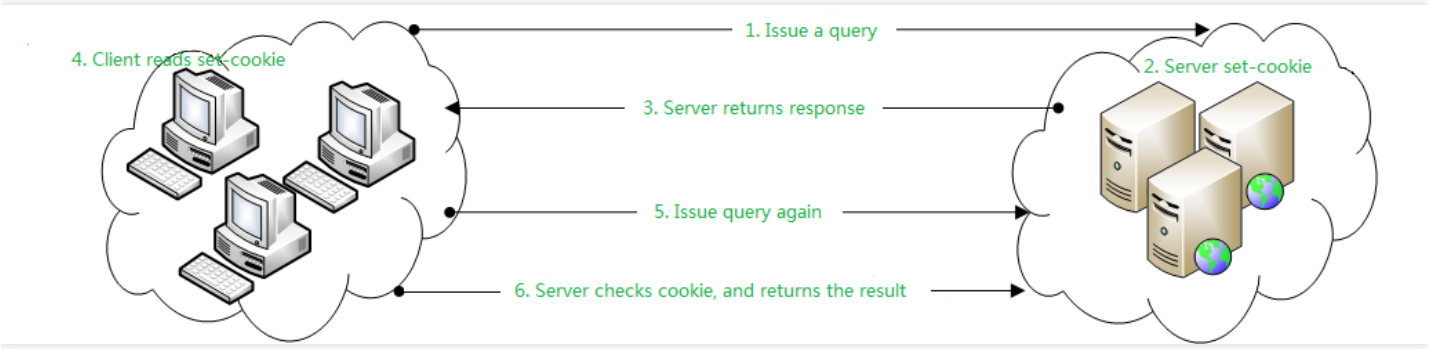
HTTP is a stateless protocol, i.e., there is no need to establish a persistent connection between the client and server. Such a connection is based on the request-response mode, where after a connection is established between the client and server, the client submits a request to the server, the server returns a response after receiving it, and then the connection is closed.

Therefore, once the client is disconnected from the server after the request is completed, they no longer have any relationship. When a user logs in on page 1 and then is redirected to page 2 of the same web application, how can page 2 know that the user has already logged in? In other words, how can the server determine that two different requests are from the same client when the client initiates another request?

Under the HTTP, the server cannot detect the relationship between different requests. To do so, a state is needed to mark each request. If the state marks of two requests are the same, then it indicates that the two requests are initiated by the same client.

A cookie is such a state bit used to mark a request. After many years of development, cookie has become more and more standardized and been widely accepted as a universal standard.

How a Cookie Works



1. When a request is initiated to Tencent Cloud for the first time, the HTTP request header is as follows:

```
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,/;q=0.8
Accept-Encoding:gzip, deflate, sdch
Accept-Language:en,zh-CN;q=0.8,zh;q=0.6
Connection:keep-alive
Host:cloud.tencent.com
```

2. After the request reaches the Tencent Cloud server, the server will generate a response and write cookie information to the response header:

```
Set-Cookie:BD_HOME=1; path=/
Set-Cookie:__bsi=14934756243064632384_00_0_I_R_174_0303_C02F_N_I_I_0; expires=Thu, 19-Nov-15 14:14:50 GMT; domain=www.qcloud; path=/
Set-Cookie:BDSVRTM=172; path=/
```

3. After receiving the response header, the client browser will write the cookie information to the local system for management.

4. When initiating another request to the server, the client will send an HTTP header containing `Cookie: name=value; name2=value2` to send the locally stored cookie. The information of the request header is as follows:

```
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,/;q=0.8
Accept-Encoding:gzip, deflate, sdch
Accept-Language:en,zh-CN;q=0.8,zh;q=0.6
Connection:keep-alive
Cookie:BD_HOME=1; BDSVRTM=0; BD_LAST_QID=15071962345319158759570 57
Host:cloud.tencent.com
```

5. After receiving the request, the server will get the cookie information from the request header, analyze the cookie, and then return a response to the client.

This is how a cookie is used to transfer information between the client and the server.

Cookie Lifecycle

A cookie has a lifecycle. When a cookie expires, it will be deleted from the client. When creating a cookie, the server can control how long the cookie can "live" on a client. A cookie's lifecycle will end in the following conditions:

- Cookies with no expiration time specified: if the server does not specify the expiration time of a cookie when creating it, the client will write the cookie in a chunk of memory provided by the browser. After the browser is closed, the chunk of memory will be released, and the cookie will be ended.
- Cookies with expiration time specified: if the server specifies the expiration time of a cookie when creating it, the cookie will be deleted upon expiration.
- When the number of cookies in the browser reaches the upper limit, the browser will delete some old cookies based on the configured rule to free up space for new ones.
- Cookies can also be deleted manually.

Managing Cookies

When creating a cookie, the server will generally specify the following two options:

- domain
- path

These two options determine the domain name and location of the created cookie.

By default, `domain` will be set to the domain name of the page that creates the cookie. When the client sends a request again to the same domain name, the cookie will be sent to the server too. If `domain` in a cookie is set to a top-level domain name, all second-level domain names under it will have the same cookie; therefore, the cookies at a top-level domain name often conflict with those at second-level domain names.

When a request is sent, the browser will make an end comparison of the `domain` value and the requested domain name (i.e., comparing the domain names starting from the end of the string) and send the matched cookie to the server.

- If `domain` is not specified, it will be the domain name of the access address by default. If the client accesses a top-level domain name, the set cookie can be shared by other second-level domain names. Therefore, operations such as login are generally performed at the top-level domain name.
- A second-level domain name can read its own cookie and the cookie whose `domain` is set to the top level, but it cannot read the cookies of another second-level domain name. In this case, to share a cookie among multiple second-level domain names, you need to set `domain` to the top-level domain name, so that all second-level domain names under it can use this cookie. It should be noted that a top-level domain name can only get the cookie whose `domain` is set to the top level and cannot get cookies of second-level domain names.

As for `path`, only when the path specified by `path` exists in the URL requested by a client will the client send the cookie message header. The `path` parameter determines the matching rule for the cookie sent from the client to the server. Generally, the value of `path` and the requested URL are compared character by character starting from the first one. If all characters are matched, the cookie message header will be sent. It should be noted that only after the `domain` value is satisfied will `path` be compared. The default value of `path` is the path in the URL corresponding to the sent message header `Set-Cookie`.

The section above summarizes cookie management in terms of browser limitations and options for cookie generation. Next, some snippets of sample code will be used to demonstrate how to create and get a cookie.

Creating a cookie on the server

The Tencent Cloud server sends an HTTP response header containing `Set-Cookie` to create a cookie as shown below:

```
// Create a cookie object
Cookie co = new Cookie( “site” ,  “http://cloud.tencent.com “);
co.setDomain( “test.com” );
// Send the cookie to the client through the response header
response.addCookie(co);

Cookie co = new Cookie( “site” ,  “http://qcloud.com “);
co.setDomain( “test.com” );
co.setPath( “/pages” );
co.setMaxAge(3600); // Unit: seconds
co.setHttpOnly(true);
co.setSecure(false);
response.addCookie(co);
```

Reading a cookie by the client

When the client initiates a request to the server, if `domain` and `path` are matched successfully, it will send the corresponding cookie together to the server. If there are too many cookies under a `path`, the HTTP request header may be too long. After the request arrives at the server, the cookies can be read as shown below:

```
Cookie[] cookies = request.getCookies();
if (cookies != null) {
for (int i = 0; i < cookies.length; ++i) {
// Get the specific cookie
Cookie cookie = cookies[i];
// Get the cookie name
```

```
String name = cookie.getName();
String value = cookie.getValue();
out.print("Cookie name:" + name + " Cookie value:" + value + "
" );
}
}
```

HTTP Status Codes

Last updated : 2020-05-12 16:18:57

An HTTP status code is a 3-digit code representing the HTTP response status of the webpage server. Such codes are defined by the RFC 2616 standard and further expanded by standards such as RFC 2518, RFC 2817, RFC 2295, RFC 2774, and RFC 4918.

The first digit of a status code represents one of the following five response classes:

1xx: information response, indicating that the request has been received and needs to be further processed
2xx: success response, indicating that the action was successfully received, understood, and accepted
3xx: redirection response, indicating that further action must be taken in order to complete the request
4xx: client error, indicating that the request contains syntax errors or cannot be executed correctly
5xx: server error, indicating that the server cannot properly run a correct request

Below are descriptions of common response return values:

100 - The client must continue to send the request
101 - The client asks the server to switch HTTP protocol versions as requested

200 - The transaction succeeded
201 - The URL of the new file has been known
202 - The request has been accepted for processing, but the processing has not been completed
203 - The returned message is indefinite or incomplete
204 - The request has been received, but the returned message is empty
205 - The server has completed the request, but the user proxy must reset the browsed files
206 - The server has partially completed the user's GET request

300 - The requested resource can be obtained in multiple locations
301 - The request data was deleted
302 - The requested data was found at another address
303 - The client is recommended to visit another URL or use another access method
304 - The client has executed GET, but the file remains unchanged
305 - The requested resource must be obtained from the address specified by the server
306 - This code was used in the previous version of HTTP and has been disused
307 - The requested resource was temporarily deleted

400 - Bad request such as syntax error
401 - The request authorization failed
402 - The valid `ChargeTo` header response is retained
403 - The request is forbidden
404 - No file, query, or URL was found
405 - The method defined by the user in the `Request-Line` field is not allowed
406 - Not acceptable. The requested resource is inaccessible
407 - Similar to 401. The user must be authorized first on the proxy server
408 - The client failed to complete the request within the time specified by the user
409 - The request cannot be completed in the current resource status
410 - This resource is no longer available on the server and there is no reference address
411 - The server rejected the user-defined `Content-Length` attribute request
412 - One or more request header fields are incorrect in the current request
413 - The requested resource is larger than the size allowed by the server
414 - The requested resource URL is longer than the length allowed by the server
415 - The format of the request project is not supported by the requested resource
416 - The request contains the `Range` request header field, but there is no range indication value within the current requested resource, and the request does not contain the `If-Range` request header field
417 - The server does not satisfy the expected value specified in the `Expect` header field of the request. If it is a proxy server, it is possible that the server at the next level cannot fulfill the request

500 - An internal server error occurred
501 - The requested function is not supported by the server
502 - The server is temporarily unavailable, sometimes to prevent system overload
503 - The server is overloaded or down for maintenance
504 - The gateway is overloaded, and the server uses another gateway or service to respond to the user, but the set waiting time is too long
505 - The server does not support or refuses to support the HTTP version specified in the request header

SSL Certificate Chain

Last updated : 2020-03-09 15:03:28

Definition of SSL certificate chain

There are two types of certificate authorities (CAs): root CAs and intermediate CAs. For an SSL certificate to be trusted, it must be issued by a CA included in the trusted store connected to by the device.

If the certificate is not issued by a trusted CA, the connecting device (e.g., a web browser) will check whether the certificate is issued by a trusted CA until no trusted CA can be found.

The list of SSL certificates goes from root certificate to intermediate certificate and then to end-user certificate.

Example of SSL certificate chain

Assume that you purchase a certificate from Qcloud CA and the domain name is `example.qcloud` .

Qcloud is not a root certificate authority. In other words, its certificate is not directly embedded in your web browser and cannot be explicitly trusted.

- Qcloud utilizes a certificate issued by Alpha, an intermediate Qcloud CA.
- Alpha utilizes a certificate issued by Beta, an intermediate Qcloud CA.
- Beta utilizes a certificate issued by Gamma, an intermediate Qcloud CA.
- Gamma utilizes a certificate issued by the Root of Qcloud.
- The Root of Qcloud is a root CA. Its certificate is directly embedded in your web browser and can be explicitly trusted.

In the above example, SSL certificate chain is represented by 6 certificates:

1. End-user certificate: issued to `example.qcloud` by Qcloud CA.
2. Intermediate certificate 1: issued to `example.qcloud` by Alpha, an intermediate Qcloud CA.
3. Intermediate certificate 2: issued to Alpha by Beta, an intermediate Qcloud CA.
4. Intermediate certificate 3: issued to Beta by Gamma, an intermediate Qcloud CA.
5. Intermediate certificate 4: issued to Gamma by the Root of Qcloud.
6. Root certificate: issued by and to the Root of Qcloud.

Certificate 1 is called end-user certificate, certificates 2–5 are called intermediate certificates, and certificate 6 is called root certificate.

When you install your end-user certificate `example.qcloud`, you must bundle all intermediate certificates and install them along with the end-user certificate. If the SSL certificate chain is invalid or broken, your certificate will no longer be trusted by some devices.

SSL Mutual/Unidirectional Authentication

Last updated : 2020-03-09 14:59:42

Specific process of SSL mutual authentication

- The browser sends a request to connect to the secure server.
- The server sends its own certificate and related information to the client browser.
- The client browser checks whether the certificate sent from the server is issued by a trusted CA. If yes, it will execute the protocol. Otherwise, it will send a warning message to the client that the certificate is untrustworthy and ask the client whether to proceed.
- The client browser compares messages in the certificate, such as domain name and public key, with messages sent from the server. If they are the same, the client browser will confirm the authenticity of the server.
- The server sends a request to the client for its certificate, and verifies the certificate after receiving it. If the verification fails, the connection will be rejected. Otherwise, the server will get the client's public key.
- The client browser tells the server its supported symmetric password schemes for communication.
- The server selects the password scheme with the highest encryption level from the schemes sent from the client, and notifies the browser after encrypting it with the client's public key.
- The browser selects a session key for this password scheme, and sends it to the server after encrypting it with the server's public key.
- The server receives the message from the browser, and decrypts it with its private key to get the session key.
- The server and the browser use the symmetric password scheme for subsequent communication where the symmetric key is encrypted.

Mutual authentication requires the server and client to provide identity verification, which improves security as only users authorized by the server can access.

Specific process of SSL one-way authentication

- The client browser sends the server the version number of its SSL protocol, the type of encryption algorithm, the random number generated, and other relevant information needed for communication.

- The server sends the client the version number of its SSL protocol, the type of encryption algorithm, the random number and other relevant information, as well as its certificate.
- The client uses the information sent from the server to authenticate the server, including whether its certificate has expired, whether the CA issuing the certificate is trustworthy, whether the public key of the CA certificate can correctly decrypt the CA's digital signature, and whether the domain name in the server certificate is the same as the actual domain name of the server. If the authentication fails, the communication will be disconnected. Otherwise, the client will proceed to step 4.
- The client randomly generates a "symmetric password" for subsequent communication, encrypts it with the server's public key obtained from the server certificate in step 2, and then sends the encrypted "pre-master password" to the server.
- If the server requires client authentication (optional during handshake), the client can create a random number and sign the data, and send the signed random number along with its own certificate and the encrypted "pre-master password" to the server.
- If the server requires identity verification, it must check the validity of the client certificate and the signed random number, including the validity period of the client certificate, whether the CA issuing the certificate is trustworthy, whether the public key of the CA can correctly decrypt the digital signature of the CA issuing the client certificate, and whether the certificate is in the Certificate Revocation List (CRL). If the verification fails, the communication will be disconnected immediately. Otherwise, the server will use its own private key to decrypt the encrypted "pre-master password" and then take a series of steps to generate a master password (the same password will also be generated by the client).
- The server and the client use the same master password, which is called "session password", and a symmetric key is used for encrypting and decrypting secure data communication of SSL protocol. The integrity of data communication should be guaranteed during SSL communication to prevent changes.
- The client sends a message to the server, indicating the master password generated in the previous step will be used as the symmetric key in the subsequent data communication. It also informs the server that handshake on the client side has ended.
- The server sends a message to the client, indicating the master password generated in the previous step will be used as the symmetric key in subsequent data communication. It also informs the client that handshake on the server side has ended.
- As SSL handshake ends, SSL data communication begins, i.e., the client and the server start to use the same symmetric key for data communication while checking the integrity of communication.

SSL one-way authentication only requires that an SSL certificate is installed on a website, which can be accessed by any user (except restricted IPs, etc.). Only the server provides identity verification.

Difference between SSL mutual authentication and one-way authentication

SSL mutual authentication requires both the server and client have certificates, while SSL one-way authentication does not require a client CA certificate.

Compared with mutual authentication, one-way authentication does not need server-side verification of the client certificate. When negotiating a symmetric password scheme and symmetric session key, the password scheme sent from the server to the client is not encrypted (does not affect the security of SSL process). The specific communication content is the encrypted data. In the event of third-party attacks, only the encrypted data will be obtained. To get any useful information, the third party has to decrypt the encrypted data, whose security depends on the security of the password scheme. Currently, a password scheme is secure as long as the communication key is long enough. This is why 128-bit encryption is used for communication.

SSL one-way authentication can be configured for general web applications. However, identity verification may be required for client to connect to financial applications. SSL mutual authentication should be used.