

Introduction

ShadowQUIC is 0-RTT QUIC based proxy with SNI camouflage.

Shadowquic doesn't provide any authentication layer. Authentication is provided by JLS protocol.

Command

Each proxy request is started with a command carried by a bistream. Only client can initiate a command.

There are several types of command indicated by CMD field:

- 0x01 : TCP Connect

CMD	SOCKSADDR
1	Variable

- 0x03 : UDP Association over datagram extension

CMD	SOCKSADDR
1	Variable

- The SOCKSADDR carries the binding address indicating which address and port client is listening on the remote. It is **NOT** the destination address.

- 0x04 : UDP Association over unistream

CMD	SOCKSADDR
1	Variable

- 0x05 : SunnyQUIC authentication

CMD	AUTH_HASH
1	64

- 0xFF : Customized Extensions

- User can customize protocol by adding 8 byte new opcode as subcommand

CMD	OPCODE
1	8

SOCKSADDR field is socks address format:

ATYP	ADDR	PORT
1	Variable	2

- ATYP address type of following address
 - IP V4 address: 0x01
 - DOMAINNAME: 0x03
 - IP V6 address: 0x04
- ADDR desired destination address
- PORT desired destination port in network octet order

TCP Proxy

TCP Connect command is supported to proxy forward TCP connection.

TCP Connect

TCP proxy task is directed followed by TCP Connect command like *socks5*

UDP Proxy

The UDP proxy scheme is greatly different from common protocol like TUIC/hysteria. The principle of design is to decrease datagram header size and reaches the **maximum MTU size**.

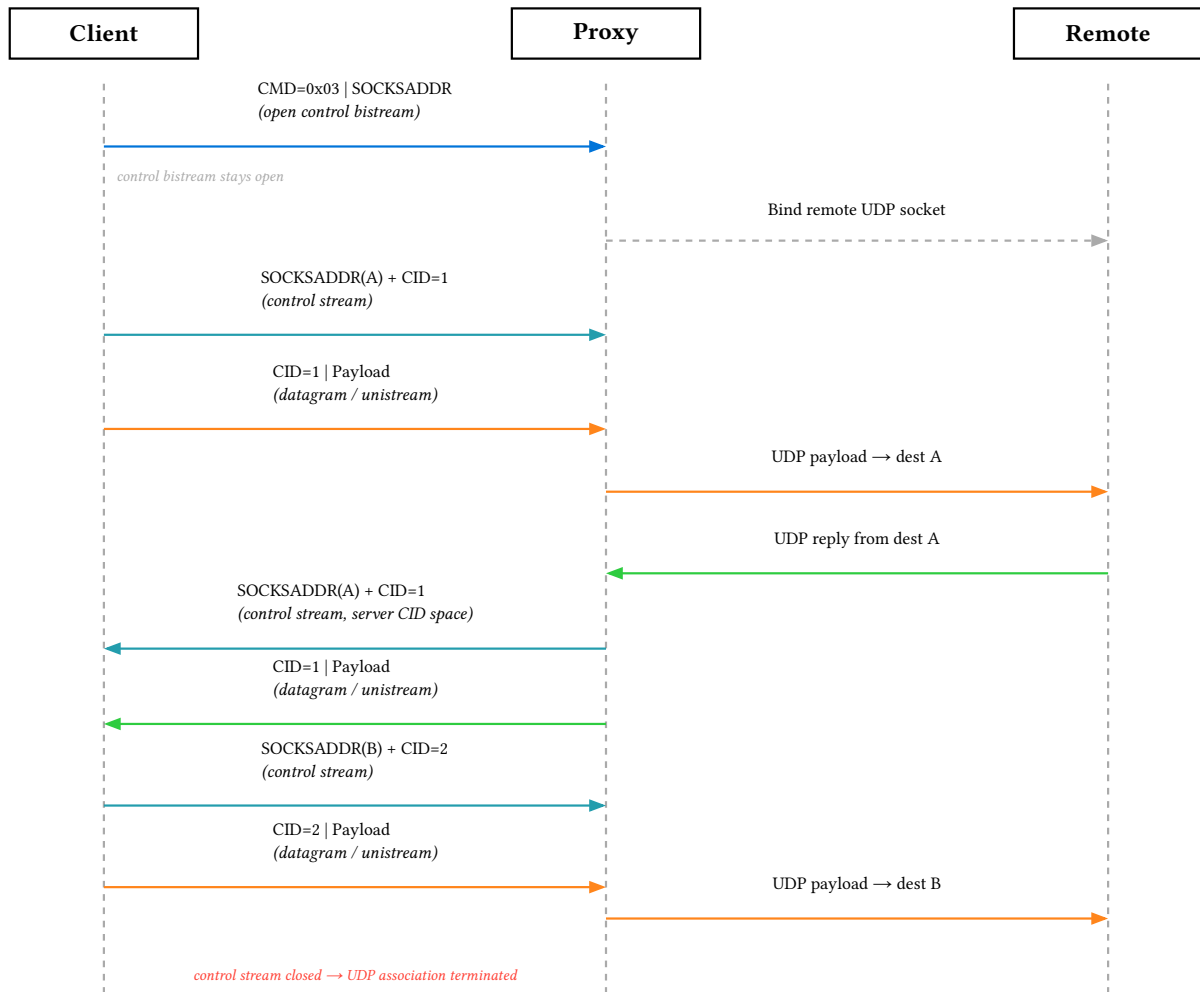


Figure 1: UDP Association data transfer process. CID = 1 means Context ID = 1

The design has heavily considered RFC 9298

SOCKSADDR	CONTEXT ID
Variable	2

UDP Associate command is carried by bistream called **control stream**. For each datagram received from local socket or remote socket a control header consists of SOCKSADDR and CONTEXT ID is sent. This

control header is sent at least once for each new CONTEXT ID. The CONTEXT ID is used to identify the datagram stream of this destination (even for receiving the returning packet of destination server).

When receiving datagram from destination, the destination address must be the same as sending address. If outgoing packet is domain address type, the receiving packet must use domain as destination too. Resolved IP address is **NOT** allowed.

For each connection, implementation must maintain two CONTEXT ID spaces. One is for client to server direction. The other is for server to client direction. These two id spaces are independent. The header of both direction is sent in the **same** control bistream.

control stream doesn't send payload. The payload is carried by unistream or datagram extension chosen by user. Control stream **MUST** remain alive during udp association task.

If control stream is terminated, the udp association task **must** also be terminated.

UDP Associate command associates a remote socket to local socket. For each destination from a local socket the datagram will be assigned a CONTEXT ID which is in **one-to-one correspondance** to four tuple: source udp ip:port and destination udp ip:port.

Each datagram payload will be prepended with a 2 bytes context ID.

For each datagram from local socket or remote socket the SOCKSADDR and CONTEXT ID pair will be sent in the control stream. And SOCKSADDR and CONTEXT ID pair will be sent **at least once** for each new CONTEXT ID.

Associate Over Stream

CONTEXT ID	LEN	PAYLOAD	LEN	PAYLOAD	...
2	2	Variable	2	Variable	...

If the datagram is carried by QUIC unistream, a 2 byte length tag is prepended to the payload. For the following datagram with the same context id, unistream could be reused, and context id is not needed to be sent. Only LEN field and PAYLOAD will be sent. Namely for each unistream, CONTEXT ID is sent only once right after this stream is opened,

Associate Over Datagram

CONTEXT ID	PAYLOAD
2	Variable

If datagrams are carried by QUIC datagram extension, the payload is sent directly without length field (only with Context ID).

SunnyQUIC

SunnyQUIC is the twin protocol of **ShadowQUIC**. It is nearly the same as **ShadowQUIC**. The only difference is that **SunnyQUIC** gives up JLS layer and provide a QUIC layer authentication. The underlying connection is native QUIC connection.

Authentication

SunnyQUIC adds a new *authentication* command. The command is carried by the bistream.

CMD	AUTH_HASH
1	64

The CMD field is 0x5 for authentication command, The AUTH_HASH field is truncated 64byte hash: SHA256(username:password)[0..64]

For client, the authentication command can be issued in parallel with other proxy commands.

For server, it should block any commands until authentication is finished. If authentication fails, server should terminate the connection.

Custom commands

In order to provide flexibility, we define 0xFF to be *customized extensions*. Users can define their own subcommand to extend protocol usage.

Subcommand OPCODE

Here is the list of supported subcommands

- Conn(0x1): connection subcommand, used to deal with per connection operation.
- User(0x2): user management subcommand, used to deal with per user operation.

Conn Subcommand

Conn subcommand can accept one byte parameter, Here 0x0 means GetConnStats. The stream will return QUIC connection stats in this bistream.

CMD(0xff)	OPCODE(0x1)	GetConnStats(0x0)
1	8	1

The response can be expressed by RUST data struct, `Result<ConnStats, ExtError>`:

```
pub struct ConnStats {
    /// The length of the this data, 8+8+8+2=26
    pub len: u32,
    pub lost_packets: u64,
    pub sent_packets: u64,
    /// In unit of milliseconds
    pub rtt: f64,
    pub current_mtu: u16,
}
/// 1 byte tag for enum variant
#[repr(u8)]
pub enum SQExtError {
    NotAvailable = 0x0,
}

/// 1 byte tag for enum variant
#[repr(u8)]
pub enum Result<ConnStats, ExtError> {
    Ok(ConnStats)=0x0,
    Err(ExtError)=0x1,
}
```

Note that the length of ConnStats is encoded in u32 and prepended to the message content. The length field is mainly for future compatibility

Here is the serialized bytes stream of response `Ok(ConnStats)`

Ok(0x0)	len(0x1a)	lost_packets	sent_packets	rtt	current_mtu
1	4	8	8	8	2

User Subcommand

User subcommand is used for runtime user management and per-user statistics. The subcommand is only available to authenticated users whose username starts with admin.

All User requests use the following prefix:

CMD(0xff)	OPCODE(0x2)	USER OPCODE	PAYLOAD
1	8	1	Variable

The supported USER OPCODE values are:

Opcode	Name	Payload	Response
0x0	AddUser	AuthUser	Result<(), SQExtError>
0x1	RemoveUser	UserName	Result<(), SQExtError>
0x2	ListUsers	Empty	Result<Vec<UserName>, SQExtError>
0x3	GetUserStats	UserName	Result<UserStats, SQExtError>
0x4	KillUserConn	UserName	Result<(), SQExtError>
0x5	GetAllStats	Empty	Result<Vec<UserStats>, SQExtError>

AuthUser contains two strings:

USERNAME_LEN	USERNAME	PASSWORD_LEN	PASSWORD
4	Variable	4	Variable

UserName is encoded as a string:

LEN	UTF-8 BYTES
4	Variable

Vec<T> is encoded as a 4 byte item count followed by each encoded item:

COUNT	ITEM 0	ITEM 1	...
4	Variable	Variable	...

The response uses the same one byte Result tag as Conn subcommand: Ok is 0x0 and Err is 0x1. Result<(), SQExtError> successful response therefore contains only one byte 0x0.

SQExtError is encoded as:

Tag	Name	Payload
0x0	NotAvailable	Empty
0x1	PermissionDenied	Empty
0x2	NotFound	Empty
0xff	Other	String

When the caller is not an admin user, the server returns Err(PermissionDenied). When the server has no user manager implementation, the server returns Err(NotAvailable). Removing, querying stats for, or killing connections of a missing user returns Err(NotFound).

UserStats is a size-tagged struct. A 4 byte payload length is prepended for future compatibility. The current payload length is 56 + username byte length.

LEN	tcp_sent	tcp_recv	udp_sent	udp_recv	tcp_conns	udp_conns	conn_num	username
4	8	8	8	8	8	8	4	String MSG LEN

Here is the serialized bytes stream of response `Ok(UserStats)`:

0k(0x0)	LEN	tcp_sent	tcp_recv	udp_sent
1	4	8	8	8
udp_recv	tcp_conns	udp_conns	conn_num	username
8	8	8	4	String MSG LEN

All integers and f64 values are encoded in network byte order.