Extended Abstract: Oscur0: One-Shot Circumvention without Registration

Mingye Chen
University of Michigan
mingyech@umich.edu

Jack Wampler
University of Colorado Boulder
Jack.Wampler@colorado.edu

Abdulrahman Alaraj
University of Colorado Boulder
abal6272@colorado.edu

Gaukas Wang
University of Colorado Boulder
Gaukas.Wang@colorado.edu

Eric Wustrow
University of Colorado Boulder
ewust@colorado.edu

ABSTRACT

Some proxies such as Conjure or Snowflake require users to register before their client can connect to an agreed-upon proxy's IP address. Registration adds additional latency, and provides an alternative avenue for censors to block to prohibit access to the proxy.

In this paper, we detail a proxy design that removes the need for registration before connecting, by leveraging UDP protocols such as DTLS or QUIC and encoding data directly in the first packet to the proxy's IP. This is not possible in TCP-based protocols, since the station-based proxy would not know to respond to an initial SYN packet, but UDP-based protocols can carry data in the first packet, allowing us to signal the proxy. Our system, `Oscur0`, is designed to work within a Refraction Networking context, and makes connecting faster and less vulnerable to blocking than existing Refraction schemes.

We outline our basic design, and detail several challenges with UDP, both for circumventors building UDP-based transports and for censors trying to block them alike. We implement a proof-of-concept using DTLS, and discuss how similar strategies could be applied to other UDP protocols such as QUIC. Our scheme provides a promising direction for Refraction Networking transports, and we believe they could be applicable to broader classes of circumvention protocols as well.

KEYWORDS

censorship circumvention, DTLS, refraction networking, UDP, DPI

1 INTRODUCTION

Refraction Networking is a censorship circumvention technique that places proxies inside the network, allowing clients to leverage unused IPs or decoy sites as proxies so long as connections pass by the proxy's in-network station. These proxies are harder for censors to block by IP address, because any IP that passes by the station could be used as a proxy.

However, some Refraction schemes require the client to **register** before they are able to use the proxy. For instance Conjure [7], the most recently-deployed Refraction protocol [13], has clients first send a short registration message to the in-network station to signal that the client intends to connect to an unused ("phantom") IP address. Then, the client can make a TCP connection and the station will pick up the other end, effectively "conjuring" a proxy at the phantom IP. Registration can occur using several channels, including domain-fronted HTTP endpoints, DNS messages, or by using other more restrictive Refraction schemes such as TapDance [14] to communicate the shared secret, phantom IP, and other parameters needed for the station to know to respond to the client's TCP SYN packet sent to the phantom address.

This two-stage process (register, then connect) has several downsides. First, censors can detect and block either registration or subsequent phantom connections, effectively creating single-points of failure for the proxy. Similarly, registration methods may be unreliable or could become unavailable for reasons independent of censorship; for example domain-fronting has become harder to use as more CDNs have moved to ban it [1, 2]. Finally, requiring registration adds extra latency to connections: Conjure clients currently wait several seconds after registering for their registration to propagate from the registration servers to Conjure stations before the client can connect to the phantom IP.

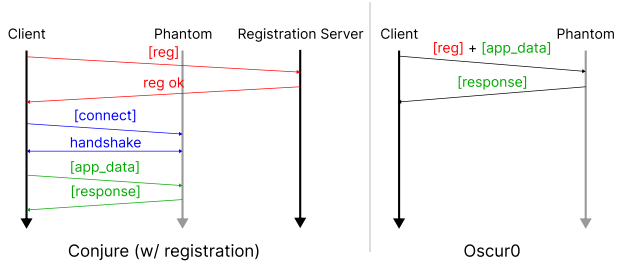

Figure 1: A high-level comparison between refraction transports using TCP and our presented UDP alternative solution.

In this work we outline `Oscur0`[1], a new Conjure-like Refraction Networking scheme that uses UDP to remove the need for a separate registration mechanism. `Oscur0` works by sending a DTLS [11] (or optionally QUIC [5]) packet to a phantom IP, steganographically encoding within it the necessary keys and parameters needed to signal to an observing station that the client wants to be proxied. Because `Oscur0` uses UDP, the client can send data within the first packet to the phantom IP. Contrasted with TCP, the client could

[1] One-Shot Circumvention Unencumbered by Registration using 0-RTT

Free and Open Communications on the Internet (1), 32–34

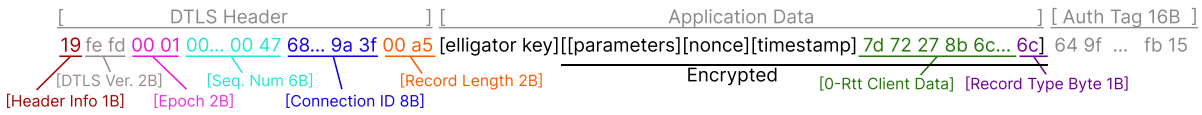

Figure 2: Example `Oscur0` connection establishment packet using a DTLS 1.2 connection ID packet structure.

only send an initial TCP SYN packet (sans data), and without the station knowing if this is a client (via registration), the station won't know if it should respond with a SYN-ACK. In `Oscur0`, the station can see the client's first packet and derive a shared secret with the client, and respond as the phantom.

`Oscur0` reduces the connection establishment time by several round-trips, obviating the need for registration, TCP handshakes, and even application-level handshakes that would otherwise be present in a typical Conjure connection as show in figure 1. This reduces the latency, and removes the potential for a censor to block the scheme by blocking registration alone. In addition, by using a UDP-based transport, `Oscur0` connections can more easily blend into peer-to-peer traffic, as DTLS is commonly used in protocols such as WebRTC.

Related work. One of the currently most widely used UDP based censorship circumvention protocols is Snowflake which builds upon WebRTC [6]. WebRTC is a peer-to-peer (P2P) protocol built on top of DTLS, but also requires a registration phase through a trusted third party. Once past the negotiation phase the connection establishes a normal DTLS connection. Like Conjure, Snowflake could also benefit from `Oscur0`-style connections to simplify registration. Appendix A shows how this could be done.

2 THE CASE FOR UDP

In Conjure, the purpose of registration is for the client to 1) signal that it wants to connect to a phantom IP, 2) establish a shared secret with the station, and 3) provide other data such as a covert proxy destination and transport parameters. To remove registration and communicate with a phantom IP in the first step, we must still share the above information with the station. Thus, the client must send these data in the first packet so the station can recognize if the connection is valid and determine if it will respond.

Since we cannot use TCP for this, we identified DTLS and QUIC as two potential UDP-based candidates. DTLS is used by WebRTC [10] and widely used on the internet for video conferencing applications such as Microsoft Teams, Google Meet, Discord etc. QUIC is being widely adopted for HTTP/3 [4].

To send the necessary registration data, QUIC supports 0-RTT handshakes and connection migration features, and DTLS [12] supports 0-RTT session resumption. Figure 2 demonstrates how such features can be leveraged to include the registration data in the encrypted application data in the first packet.

Past transports that tunneled over VoIP or voice conferencing software were identifiable by censors due to the fingerprint of retransmitting dropped packets, a feature not characteristic of normal VoIP communications [8, 9]. However, protocols such as DTLS and QUIC, which support both fault-tolerant and sequential modes without external indicators of their operation mode, offer a way to create reliable transports for such traffic without alerting censors through retransmission.

3 DESIGN

The client first generates a private and public key pair. Using the station's public key, the client derives an ECDH shared secret with the station. We use the shared secret as the DTLS master secret, and use it to generate the state of the DTLS connection with random sequence numbers and connection ID (CID) for the client and station. We set the epoch to 1 similar to normal DTLS connections after the initial handshake [11].

When the client sends its first packet, we prepend a random 10 byte ID and registration information such as the covert address and transport parameters to the application data. The combined application data is encrypted to ciphertext normally by the DTLS library. Similar to Conjure's Tapdance-based decoy registration, we use Elligator [3] to encode the client's public key. We prepend the client's public key to the ciphertext and increment the data length in the record header by the length of the encoded public key.

The station is able to determine if a connection is a valid `Oscur0` connection by extracting an assumed client public key from the first bytes of ciphertext, deriving the ECDH shared secret, and trying to decrypt the ciphertext. If the decryption succeeds, the station extracts the necessary data (transport parameters, what IP the client wants to connect to, etc) and starts the proxy connection, responding as the phantom IP. If the decryption fails, the station drops the connection and does not respond.

Unlike data from a full handshake, this type of 0-RTT data is vulnerable to **replay attacks**. To protect against this, we add a random 10-byte ID in the encrypted application data. When the first packet is received, the station checks the ID against a list (e.g. bloom filter) to see if it has been used recently. If it has been seen, the station drops the connection and does not respond.

Starting a connection without a handshake may be unusual, even if allowed by the specification. However, we can instead have the client first send a DTLS client hello message, using the key share extension to send its public key, and the client random to send an encyrypted minimal registration. The station can complete the handshake, and the client can then send the remaining registration information as encrypted application data.

Implementation. We implemented a proof-of-concept `Oscur0` client and server in around 600 lines of Go, building on top of the `pion/dtls` library [11], and tested it from a VPS in Iran. We note that while DTLS handshakes are blocked for our Iran vantage point, `Oscur0` is able to avoid this blocking as it sends only `Application Data (with CID)` packets. In the future, we hope to integrate Oscur0 with the existing Conjure deployment, and be able to compare the performance in a real-world setting.

REFERENCES

[1] 2021. Securing our approach to domain fronting within Azure. https://www.microsoft.com/en-us/security/blog/2021/03/26/securing-our-approach-to-domain-fronting-within-azure/.

[2] 2023. Fastly to block domain fronting in February 2024. https://lists.torproject.org/pipermail/anti-censorship-team/2023-October/000328.html.

[3] Daniel J Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. 2013. Elligator: elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 967–980.

[4] Mike Bishop. 2022. HTTP/3. RFC 9114. https://doi.org/10.17487/RFC9114

[5] Martin Duke. 2023. QUIC Version 2. RFC 9369. https://doi.org/10.17487/RFC9369

[6] David Fifield, Nate Hardison, Jonathan Ellithorpe, Emily Stark, Dan Boneh, Roger Dingledine, and Phil Porras. 2012. Evading censorship with browser-based proxies. In *Privacy Enhancing Technologies: 12th International Symposium, PETS 2012, Vigo, Spain, July 11-13, 2012. Proceedings 12*. Springer, 239–258.

[7] Sergey Frolov, Jack Wampler, Sze Chuen Tan, J Alex Halderman, Nikita Borisov, and Eric Wustrow. 2019. Conjure: Summoning proxies from unused address space. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2215–2229.

[8] John Geddes, Max Schuchard, and Nicholas Hopper. 2013. Cover your acks: Pitfalls of covert channel censorship circumvention. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 361–372.

[9] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. 2013. The parrot is dead: Observing unobservable network communications. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 65–79.

[10] Randell Jesup, Salvatore Loreto, and Michael Tüxen. 2021. WebRTC Data Channels. RFC 8831. https://doi.org/10.17487/RFC8831

[11] Eric Rescorla and Nagendra Modadugu. 2012. Datagram Transport Layer Security Version 1.2. RFC 6347. https://doi.org/10.17487/RFC6347

[12] Eric Rescorla, Hannes Tschofenig, Thomas Fossati, and Achim Kraus. 2022. Connection Identifier for DTLS 1.2. RFC 9146. https://doi.org/10.17487/RFC9146

[13] Benjamin VanderSloot, Sergey Frolov, Jack Wampler, Sze Chuen Tan, Irv Simpson, Michalis Kallitsis, J Alex Halderman, Nikita Borisov, and Eric Wustrow. 2020. Running refraction networking for real. *Proceedings on Privacy Enhancing Technologies* 2020, 4 (2020).

[14] Eric Wustrow, Colleen M Swanson, and J Alex Halderman. 2014. TapDance: End-to-Middle Anticensorship without Flow Blocking. In *23rd USENIX Security Symposium (USENIX Security 14)*. 159–174.

A SNOWFLAKE

In schemes such as snowflake, the client might not know the IP address of the proxy in advance. Therefore, the client still needs a way to obtain the IP address of the proxy. `Oscur0` in this case does not eliminate the latency of obtaining the proxy IP address, but could still eliminate the need of the registration server sending registration data to the proxy server, therefore saving at least 1 round trip. In addition, the IP address of the proxy might not have the same requirements as registration and could be sent through a more distributed way such as DNS or DHT, which eliminates the single point of failure of the registration server.