Independent Submission
Internet-Draft
Intended status: Informational
Expires: 11 November 2026

V. QwQ

10 May 2026

The PRF Protocol
draft-prf-protocol-01

Abstract

This document describes the PRF protocol ("Prf is Reversed Frp"), a lightweight, binary-length-prefixed TCP relay messaging protocol. PRF enables peer-to-peer tunneling of TCP connections through a public middle relay node, originally designed for Minecraft multiplayer sessions behind NAT or firewall environments. The protocol defines three roles - Server, Client, and Middle - and a set of four wireframe message types for room registration, connection requests, worker handoff, and direct Minecraft client handshake parsing.

Status of This Memo

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

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

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

This Internet-Draft will expire on 11 November 2026.

Copyright Notice

Table of Contents

1. Introduction

PRF is a custom application-layer protocol that operates over raw TCP sockets. It is designed to facilitate TCP port forwarding between peers that cannot directly connect to each other due to network constraints, such as Network Address Translation (NAT) or firewall restrictions. A publicly accessible middle relay node brokers connections between a Server (which hosts a service) and a Client (which wishes to access that service).

The protocol does not provide encryption, authentication, or integrity protection. It is intended for use in trusted or low-risk environments where the primary concern is connectivity rather than confidentiality.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Roles

The PRF protocol defines three distinct roles:

Server (Mode 1): The party that hosts the actual service (e.g., a Minecraft server). It registers a room identifier with the Middle node and waits for incoming Client connections to be forwarded.
Client (Mode 2): The party that wishes to access the Server’s service. It sends a connection request to the Middle node specifying a room identifier.
Middle (Mode 3): A publicly accessible relay node that brokers connections between Servers and Clients. It maintains a mapping of room identifiers to registered Servers and pending Client connections.

2. Protocol Overview

PRF is a request-response protocol in which all communication flows through the Middle node. The Server registers with the Middle and then polls for pending Client connections. The Client sends a connection request to the Middle. When a Client is matched with a Server, the Middle assigns a Client Worker Identifier (CW-ID) and notifies the Server. The Server then initiates a worker connection back to the Middle to claim the Client, after which a bidirectional TCP tunnel is established.

The protocol also defines a null-type mode in which no type identifier is sent. In this mode, the Middle node inspects the raw initial bytes of the TCP stream to determine routing. One recognized payload format in this mode is the Minecraft client handshake packet, making the protocol compatible with native Minecraft clients without requiring the PRF Client tool.

3. Wire Format

3.1. Byte Order

All multi-byte numeric fields are transmitted in network byte order (big-endian).

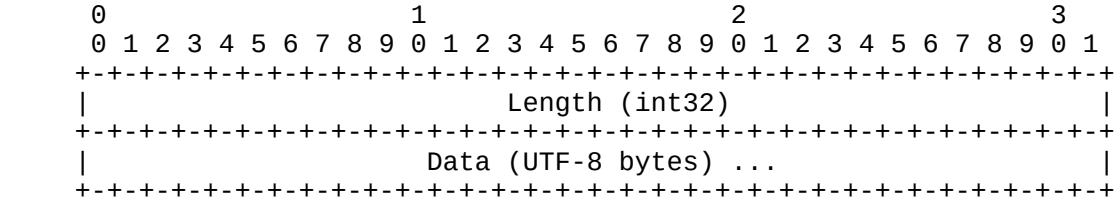
3.2. Primitive Types

3.2.1. Integer (int32)

A 32-bit signed integer encoded in 4 bytes, big-endian. The maximum value is 2,147,483,647.

3.2.2. String

A string is encoded as a length-prefixed sequence of UTF-8 bytes:



The Length field specifies the number of UTF-8 bytes that follow.

3.2.3. Byte Array (bytes)

A byte array is encoded identically to a String, with a 4-byte length prefix followed by the raw bytes. Unlike String, the payload of a byte array is not assumed to be valid UTF-8.

4. Message Types

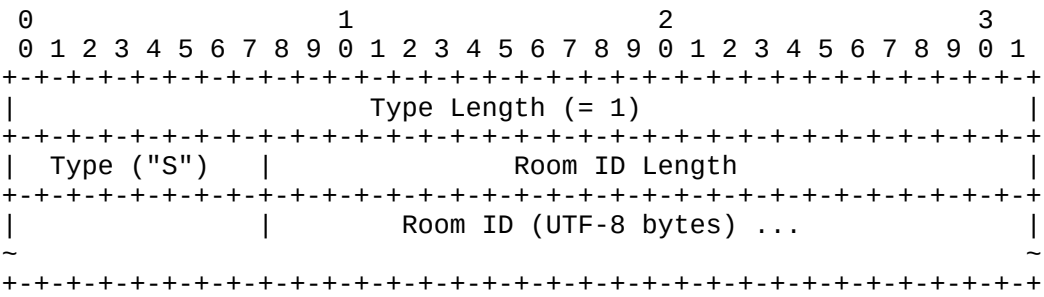
Every message sent to the Middle node begins with a type identifier field, which is a length-prefixed string containing one of the following four values. The type identifier determines how the remainder of the message is parsed.

4.1. Type "S" - Server Registration

4.1.1. Purpose

Sent by the Server to register a room identifier with the Middle node.

4.1.2. Format



The Type field is the UTF-8 string "S". The Room ID is a UTF-8 string of at most 30 bytes identifying the room.

4.1.3. Response

Upon receiving a Type "S" message, the Middle node sends a response string.

Success vs. Failure: The first character of the response string determines the outcome. A response beginning with > (U+003E) indicates success; any other response indicates failure. The remainder of the string is a server-defined, internationalized (i18n) message that MAY be displayed to the user but MUST NOT be used for protocol-level decision making.

4.1.4. Reconnection

If the Room ID ends with the character v (U+0076, case-insensitive) and the room ID is already registered, the Middle node evicts the existing Server registration and replaces it with the new connection. The evicted Server receives the string "EXIT" (without a leading type identifier) on its poll channel, indicating that it should terminate.

4.1.5. Polling

After successful registration, the Server enters a polling loop. The Middle node sends a String (without a leading type identifier) to the Server at intervals of up to 5 seconds:

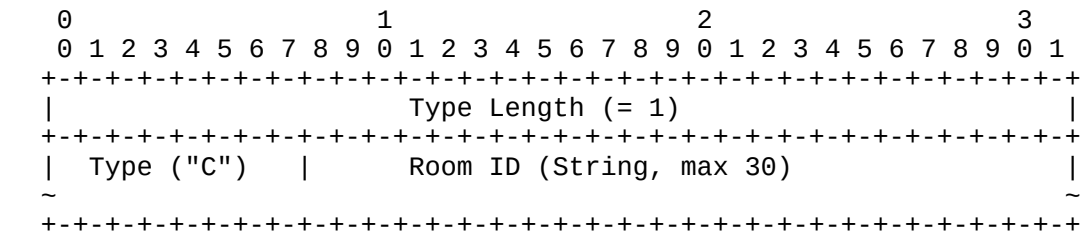
* An empty string ("") serves as a keepalive signal; the Server takes no action.
* A non-empty string is a CW-ID (Client Worker Identifier) of the form CW<n>, where <n> is a decimal integer. The Server MUST spawn a streamer to handle this Client connection by sending a Type "W" message.

4.2. Type "C" - Client Connection Request

4.2.1. Purpose

Sent by the Client to request a connection to a room identified by its Room ID.

4.2.2. Format



The Type field is the UTF-8 string "C". The Room ID is the identifier of the target room, at most 30 bytes.

4.2.3. Response

Upon receiving a Type "C" message, the Middle node sends a response string.

Success vs. Failure: The first character of the response string determines the outcome. A response beginning with > (U+003E) indicates success; any other response indicates failure. The remainder of the string is a server-defined, internationalized (i18n) message that MAY be displayed to the user but MUST NOT be used for protocol-level decision making.

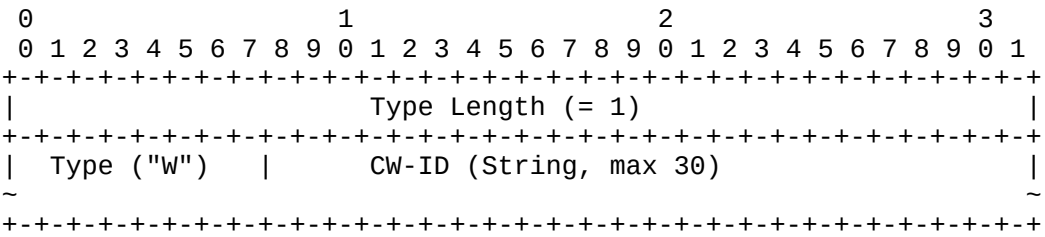
If the response indicates success, the Client SHOULD proceed to establish a bidirectional tunnel.

4.3. Type "W" - Worker Handoff

4.3.1. Purpose

Sent by the Server’s streamer to claim a pending Client connection.

4.3.2. Format



The Type field is the UTF-8 string "W". The CW-ID field is the Client Worker Identifier previously received from the Middle during polling.

4.3.3. Response

Upon receiving a Type "W" message, the Middle node sends a response string.

Success vs. Failure: The first character of the response string determines the outcome. A response beginning with > (U+003E) indicates success; any other response indicates failure. The remainder of the string is a server-defined, internationalized (i18n) message that MAY be displayed to the user but MUST NOT be used for protocol-level decision making.

On success, a bidirectional tunnel is established between this streamer’s socket and the corresponding Client’s socket. The response message MAY contain the IP address of the connected Client as informational text.

4.3.4. HAProxy PROXY Protocol Header

If the room identifier starts with the character V (U+0056, case-insensitive), the Middle node prepends a HAProxy PROXY protocol version 1 header [PROXY] to the Client-to-Server direction of the tunnel before bridging. The header has the following format:

PROXY TCP4 <client-ip> 192.0.2.1 10000 10000\r\n

Where <client-ip> is the actual IP address of the Client. The destination address and port are placeholders (192.0.2.1:10000). This mechanism allows the target service to obtain the real Client IP address.

4.4. Type null - Untyped Raw Stream Inspection

4.4.1. Purpose

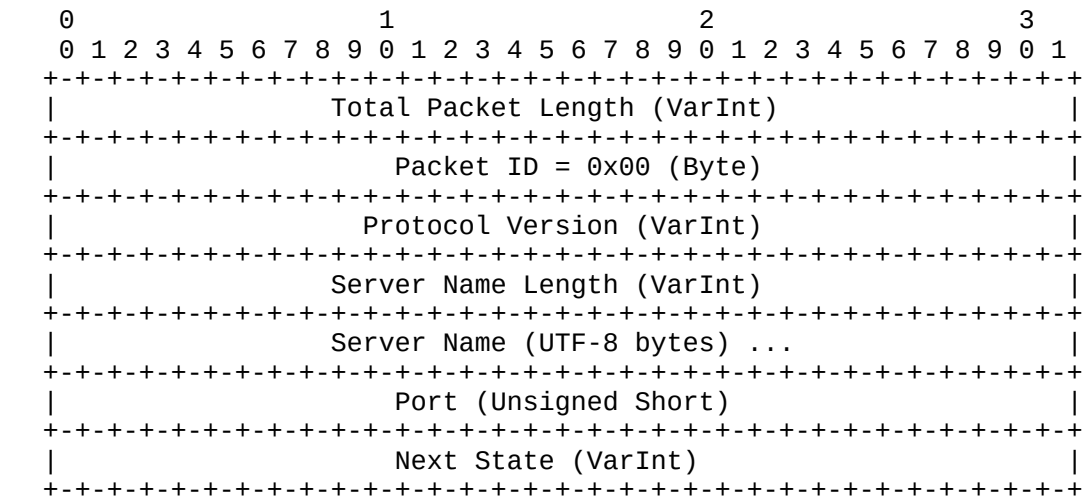
When a connection arrives at the Middle node without a leading type identifier string, the Middle enters an untyped raw stream inspection mode. Rather than rejecting the connection, the Middle reads the initial bytes from the TCP stream and attempts to identify a recognized payload format. If a format is recognized, the Middle extracts routing information (the room identifier) and bridges the connection; if not, the connection is terminated with an error.

This is not a general-purpose passthrough - it is a protocol-defined extension point that allows PRF to interoperate with existing application protocols without requiring a PRF-specific client. The specific format described below is the Minecraft client handshake, which PRF happens to be compatible with by design.

4.4.2. Recognized Format: Minecraft Client Handshake

The Minecraft client handshake packet is one payload format recognized in the null-type mode. When the Middle identifies this format, it extracts a room identifier from the Server Name Indication (SNI) hostname carried in the packet.

The packet format is defined by the Minecraft protocol. Within PRF, the relevant fields are:



The Total Packet Length MUST be at least 10 bytes and at most 100 bytes. The Packet ID MUST be 0x00 (Handshake). The Server Name Length MUST be at least 5.

4.4.3. Room ID Extraction

The Middle node extracts the subdomain from the Server Name (SNI) hostname by taking all characters before the first . (U+002E) character. For example, if the SNI is myroom.relay.example.com, the extracted room ID is myroom.

The full handshake packet is buffered and forwarded unmodified to the target Server after the tunnel is established.

4.4.4. Failure Handling

If the room ID cannot be found, the Middle node sends a Minecraft Disconnect/Login packet (Packet ID 0x00) to the Client with a JSON reason string in the format:

{"text":"<reason>"}

This terminates the Client connection with an appropriate error message displayed in the Minecraft client.

5. Protocol Constants

The following constants are defined by the PRF protocol:

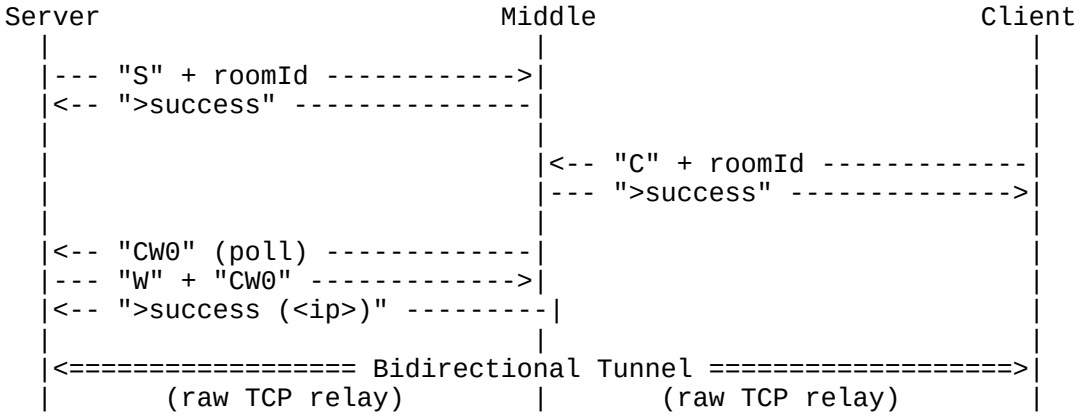
Constant	Value	Description
Maximum Room ID Length	30 bytes	Maximum UTF-8 byte length of a room/server identifier
Maximum Response Length	100 bytes	Maximum UTF-8 byte length of a Middle response string
Minimum MC Packet Length	10 bytes	Minimum Minecraft handshake packet total length
Maximum MC Packet Length	100 bytes	Maximum Minecraft handshake packet total length
Minimum Server Name Length	5 bytes	Minimum SNI hostname length
Maximum VarInt Bytes	5	Maximum bytes for a single VarInt encoding
Poll Interval	5 seconds	Interval at which the Server polls for new CW-IDs
Initial Read Timeout	5 seconds	Middle socket read timeout for type detection
Reconnect Delay	10 seconds	Delay before a Server reconnects after disconnect
Streamer Bridge Delay	10 seconds	Delay before a MiddleStreamer begins bridging data
CW-ID Prefix	"CW"	Prefix for Client Worker Identifiers
PROXY destination address	192.0.2.1	Placeholder destination IP in PROXY header
PROXY destination port	10000	Placeholder destination port in PROXY header

Table 1

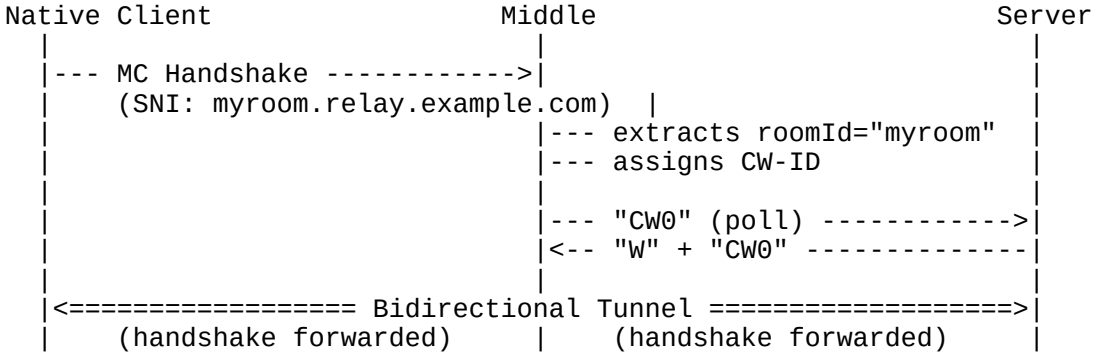
6. Protocol Flow

6.1. Server Registration and Client Connection

The following diagram illustrates the normal protocol flow for a Server registering and a Client connecting:



6.2. Untyped Raw Stream Inspection (e.g., Minecraft Client)



7. Security Considerations

The PRF protocol as defined in this document does not provide any form of encryption, authentication, or integrity protection. All data, including room identifiers, is transmitted in cleartext over TCP. The following security implications should be considered:

* *Traffic Interception:* Any party on the network path can read all tunneled data, including the room identifier and any application-layer content.

* *Traffic Injection:* Without integrity protection, an attacker can inject or modify packets in transit.
* *Room ID Enumeration:* Room identifiers are transmitted in cleartext and can be observed or guessed by an attacker, allowing unauthorized connection to a Server.
* *No Server Authentication:* Clients have no cryptographic means to verify that they are connecting to the intended Server or through a legitimate Middle node.
* *No Client Authentication:* Servers accept connections from any Client that provides the correct room identifier.

PRF is designed to provide no additional security guarantees beyond those offered by the underlying transport. The protocol itself does not perform encryption, authentication, content inspection, or access control. Consequently, the overall security of a PRF-tunneled connection depends entirely on the security properties of the application-layer protocol carried within the tunnel. Implementations and deployments SHOULD ensure that the tunneled protocol provides its own encryption and authentication if confidentiality and integrity are required.

8. IANA Considerations

This document has no IANA actions.

--- back

9. Acknowledgements

The PRF protocol name is a playful backronym: "Prf is Reversed Frp," referencing the frp (Fast Reverse Proxy) tool that inspired its design.

10. Normative References

[PROXY] HAProxy Technologies, "HAProxy Proxy Protocol", 2011, <https://www.haproxy.org/download/1.8/doc/proxy-protocol.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

Author’s Address

Venti QwQ
Email: huzpsb@qq.com