UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Discord

Redes de Computadores I

ELO 322

Integrantes:

Javier Quiroga 201421012-4

Esteban Póvez 201303071-8

Profesor:

Agustín J. González

UTFSM

Viernes 13 de Septiembre de 2019

1. Resumen

Comunicación simple y eficiente al alcance de cualquier dispositivo.

Discord es una aplicación que permite la comunicación entre pares de manera simple y eficiente, sin mucha exigencia a la plataforma en la que se está ejecutando, logrando así una comunicación efectiva a tiempo real.

A lo largo de este documento podrá observar cómo es que este programa es capaz de lograr comunicar a varios usuarios y qué procesos utiliza para ello, tomando en cuenta los protocolos usados y los puntos de acceso que se aprovecha para llevar a cabo su tarea. Se abordarán temas como protocolos de capa de transporte y procesos de traducción y direccionamiento NAT, tal como las soluciones que se encontraron para el problema de la comunicación a tiempo real (RTC) utilizando servidores web. Se espera que al terminar este documento, el lector pueda comprender un poco mejor cómo es que este programa logró solucionar de manera tan eficiente una necesidad básica que no pudo ser suplida por mucho tiempo.

2. Introducción

La necesidad de comunicarse ha sido un punto clave en la comunidad videojugadora desde que se implementó el juego en línea, pero en un inicio no se tuvo un sistema que pudiera cumplir con esta necesidad de manera simple y eficiente. En un inicio, programas como Skype o TeamSpeak intentaron suplir esta necesidad, pero lamentablemente tenían serios problemas, dado que el primero exigía mucho del procesador y muchos computadores no podían seguir el paso, y el segundo era un servicio pago con cierta complejidad que lo volvía inaccesible para quién quisiera utilizarlo sin ayuda; Aquí es cuando aparece Discord, una aplicación con una interfaz amigable con el usuario, supliendo la necesidad de comunicarse vía texto y voz a tiempo real, con la adición de diversos otros servicios como la creación de canales personales, uso de redes sociales y adición de programas de terceros con soporte de los desarrolladores para su implementación y desarrollo.

En este documento se estudiará como es que Discord logra generar una efectiva ejecución de su funcionalidad como medio de comunicación, qué protocolos utiliza y como es que se relaciona con la red que utiliza como medio.

3. Análisis del programa

La API (Application programming interface) de Discord está basada en dos capas principales, la API REST (Representational state transfer) y la API de Websockets. La Api REST es usada por Discord para operaciones generales dentro de la aplicación, principalmente enfocado en el uso de bots, los cuales pueden ser creados para Discord y realizar distintos tipos de acciones, por ejemplo, asistencia en la moderación en los canales, autentificación, reproductores de música, búsquedas en internet, etc. Por otro lado, Discord hace uso de Websockets seguros (WSS) y persistentes para el manejo de eventos en tiempo real, como solicitar conexiones, uso de identificación, persistencia de conexión, manejo de conexiones por voz, etc. En Discord la conexión a través de Websockets es llamada Gateway. Los clientes en Discord recibirán y enviaran eventos a través de la Gateway a la que están conectados y envían esta información a través de la API REST.

3.1. Gateway de Discord

Websocket es un mecanismo que proporciona un canal de comunicación bidireccional y full-duplex a través de una sola conexión TCP. Full-Dúplex se refiere a que la conexión es capaz de realizar comunicación en ambos sentidos y simultáneamente. Además de ser una conexión Full-Dúplex, otra ventaja se obtiene en si ya se estableció una conexión TCP , esta no debe iniciarse de nuevo en caso de una nueva solicitud, es una conexión persistente. En el caso de Discord Se utiliza el protocolo WebSocket Secure (wss) , junto con el protocolo criptográfico TLS 1.2 (Transport Layer Security) sobre la conexión TCP (Transmission Control Protocol) utilizando el puerto 443.

3.2. Envio

En una Gateaway , los paquetes enviados por el cliente son encapsulados en un objeto de carga útil(Gateway Payload). Este objeto puede ser serializado en un formato predefinido que puede ser binario ETF (External Term Format) o de texto plano JSON(JavaScript Object Notation), este ultimo compresible con la librería zlib. El tamaño de los paquetes (Payloads) está limitado a un máximo de 4096 bytes, superar este limite causara un termino de la conexión.

3.3. Recepción

Recibir paquetes a travez de la Gateay es mas complejo que enviar. Si se usa formato JSON con compresión, la librería debe descomprimir los paquetes antes de intentar parsearlos (Analizar sintácticamente). La Gateway no implementa un contexto de compresión compartida entre los mensajes enviados.

3.4. Conexión a Gateway

El primer paso para establecer la conexión es requerir un endpoint websocket valido, esto a travez de una llamada "Get Gateway". Lo anterior retornará una url del endpoint disponible y solo basta con acceder a ella.
Un ejemplo de URL es la siguiente "$wss://gateway.discord.gg/?v=6encoding=json$".
Ya conectado el cliente , este recibe un paquete "Hello" que contiene un tiempo en milisegundos que especifica el tiempo cada cuanto se enviarán paquetes "Heartbeat" o de latido, y se recibirán ACK por cada paquete, esto es para mantener la conexión viva entre cliente y servidor.Se terminara la conexión si no se recibe un numero predefinido de ACK, después de enviar un "Heartbeat". Luego el cliente envia un paquete "Identify" usado para gatillar el handshake inicial con la Gateway. Este contiene propiedades del cliente tales como el sistema operativo,nombre del dispositivo, clave de identificación, si soporta compresión, estado(online, no molestar,AFK, etc), entre otros.La Gateway responde a esto con un paquete "Ready" con el que finaliza el Handshake y contiene información como un identificador de sesión,en caso de reanudar conexión,información del usuario, etc.

3.5. Reconexion a Gateway

En caso de desconexiones en el canal, Discord tiene un procedimiento para reanudarlas. Cuando el cliente detecta que ha sido desconectado , este cierra la existente y abre una nueva siguiendo el procedimiento explicado anteriormente. Conectado nuevamente, el cliente envía un paquete "Gateway Resume" que contiene el identificador de sesión del paquete "Ready" y un numero de secuencia del ultimo paquete recibido y posteriormente la Gateway responde con los paquetes perdidos en orden y un ultimo paquete "Resumed" indicando que ya se enviaron todos los paquetes perdidos

3.6. Comunicación por voz

La conexión por voz es similar a la de Gateway solo que ocupa un set distinto de paquetes y una conexión separada basada en UDP(User Datagram Protocol) para la transmisión de los datos de voz.Se utiliza un puerto UDP aleatorio entre 50000 a 65535.

El primer paso para establecer comunicación por voz es informar y solicitar al servidor a través de la Gateway que se quiere establecer esta conexión, esto enviando un paquete "Gateway Voice State Update". El servidor responde a esta solicitud con dos paquetes, el primero," Voice State Update" contiene un identificador de sesión, y el segundo, "Voice Server Update" contiene la información del servidor de voz, como su dirección (endpoint). Luego de conectar al endpoint se envía un paquete "Identify" para iniciar el handshake para la conexión del Websocket de voz, y se recibe por parte del servidor el paquete "Ready" que contiene la dirección IP y puerto para realizar la conexión por UDP y también contiene el método de encriptación para los mensajes de voz. De igual manera tiene que existir un "Heartbeat" para mantener persistente la conexión del Websocket . Luego de haber las propiedades del servidor de voz UDP del paquete "Ready" , es posible realizar la conexión por voz. Debido a que UDP es usado tanto para transmitir como recibir datos, el cliente debe capaz de recibir estos paquetes incluso a través de NAT(Network address translation),el cual enmascara la IP y puerto, es por eso que Discord hace un procedimiento llamado "descubrimiento de IP", en el cual el cliente envía un paquete al servidor y este le responde con la IP y puerto externo para realizar la conexión. Con esto ya solucionado, se envía un paquete "Select Protocol" por el Websocket con la IP, puerto, protocolo usado(UDP) , y el método de encriptación deseado(actualmente Discord solo soporta el método $xsalsa20$). Finalmente, el servidor envía el paquete "Session Description" que contiene una clave secreta para la encriptación y así poder hablar por voz de manera segura en la conexión UDP establecida. En la siguiente imagen se muestra el procedimiento anteriormente explicado. Los mensajes de voz son codificados con el códec Opus, usando dos canales estéreos a una taza de muestreo de 48 kHz.

4. Resultado de parte práctica

En la aplicacion de escritorio de discord es posible acceder a la consola a travez del comando $ctrl + shift + i$. Al inicio del programa este conecta automáticamente a la Gateway como se muestra a continuación.
Se observan algunos de los pasos mencionados en la conexión a Gateway, como la respuesta al GetGateway con la URL WSS, el paquete "$Hello$" con el Heartbeat de 41.25 segundos.

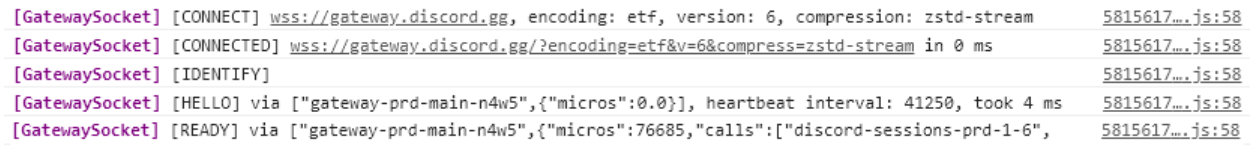

Figura 1: Conexión a Gateway en consola Discord

Por otro lado al conectar a un servidor de voz se obtiene lo siguiente.

Figura 2: Conexión a voz en consola Discord

Se ve primero que se hace una conexión a un Websocket de voz ubicado en Brasil, con un heartbeat de 13.75 segundos. Posteriormente se realiza un descubrimiento IP, para luego conectar a la dirección IP y puerto del servidor (185.50.107.219:50740) con el cliente (190.45.133.44:60668). Notar que ambos puertos están en el rango [50000-65535].

Al utilizar Wireshark mientras se probaban distintos tipos de comunicación, se pudo observar como es que los distintos protocolos actuaban según fuera necesario.

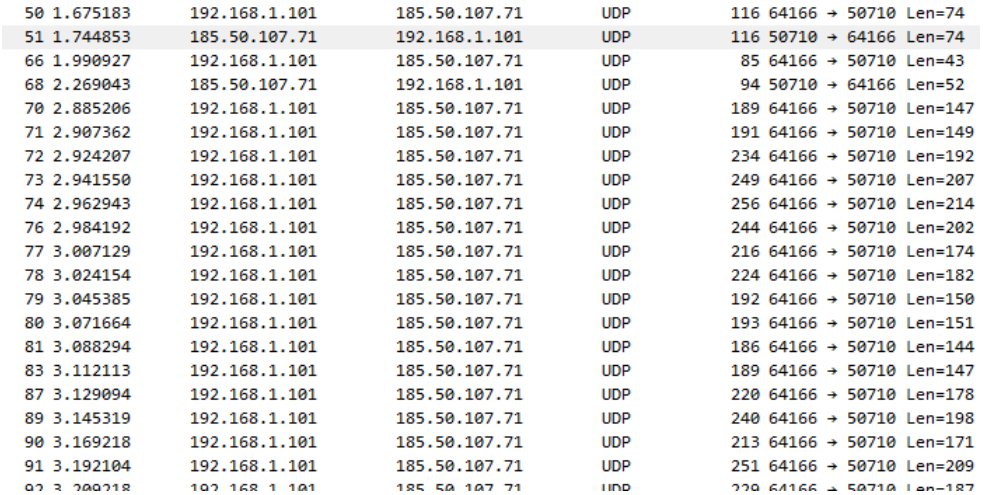

Figura 3: Protocolo capa de transporte UDP

Cuando se envian mensajes del tipo voz, se pueden observar paquetes usando el protocolo UDP (como se muestra en la figura 4). En este caso, la dirección IP correspondiente a nuestro

computador personal es 185.50.107.71, mientras que la otra dirección corresponde al servidor que conecta a los distintos usuarios utilizando el servicio. Se puede también observar, como se mencionó anteriormente, que cuando se comunica con nuestra dirección se utiliza el puerto 50710, el cual está dentro del rango estipulado (50000-65535). Estos paquetes corresponden a paquetes con la información que luego será transformada a sonido, junto a paquetes del tipo "speaking".

25	2.859782	10.2.43.251	162.159.134.233	TCP	66 49870 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
26	2.863103	162.159.134.233	10.2.43.251	TCP	66 443 → 49870 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM=1 WS=2
27	2.863158	10.2.43.251	162.159.134.233	TCP	54 49870 → 443 [ACK] Seq=1 Ack=1 Win=65536 Len=0
29	2.866706	162.159.134.233	10.2.43.251	TCP	60 443 → 49870 [ACK] Seq=1 Ack=518 Win=6432 Len=0
35	2.871536	162.159.134.233	10.2.43.251	TCP	60 443 → 49870 [ACK] Seq=360 Ack=1339 Win=7622 Len=0
39	2.915417	10.2.43.251	162.159.134.233	TCP	54 49870 → 443 [ACK] Seq=1463 Ack=451 Win=65024 Len=0
45	3.084371	10.2.43.251	162.159.134.233	TCP	54 49870 → 443 [ACK] Seq=1463 Ack=1239 Win=64256 Len=0
48	3.127532	10.2.43.251	162.159.134.233	TCP	54 49870 → 443 [ACK] Seq=1463 Ack=1269 Win=64256 Len=0
57	4.356537	162.159.134.233	10.2.43.251	TCP	60 443 → 49870 [ACK] Seq=1269 Ack=1644 Win=7622 Len=0
60	4.536163	10.2.43.251	162.159.134.233	TCP	54 49870 → 443 [ACK] Seq=1644 Ack=1655 Win=65536 Len=0
64	4.579419	10.2.43.251	162.159.134.233	TCP	54 49870 → 443 [ACK] Seq=1644 Ack=1685 Win=65536 Len=0
98	5.341855	162.159.134.233	10.2.43.251	TCP	60 443 → 49870 [ACK] Seq=1685 Ack=1734 Win=7622 Len=0
99	5.341856	162.159.134.233	10.2.43.251	TCP	60 443 → 49870 [ACK] Seq=1685 Ack=1822 Win=7622 Len=0
112	5.548156	10.2.43.251	162.159.134.233	TCP	54 49870 → 443 [ACK] Seq=1822 Ack=2069 Win=65280 Len=0
115	5.591483	10.2.43.251	162.159.134.233	TCP	54 49870 → 443 [ACK] Seq=1822 Ack=2099 Win=65024 Len=0
128	6.580285	162.159.134.233	10.2.43.251	TCP	60 443 → 49870 [ACK] Seq=2099 Ack=1915 Win=7622 Len=0
129	6.580286	162.159.134.233	10.2.43.251	TCP	60 443 → 49870 [ACK] Seq=2099 Ack=2005 Win=7622 Len=0
134	6.875956	10.2.43.251	162.159.134.233	TCP	54 49870 → 443 [ACK] Seq=2005 Ack=2483 Win=64768 Len=0
136	6.919521	10.2.43.251	162.159.134.233	TCP	54 49870 → 443 [ACK] Seq=2005 Ack=2513 Win=64768 Len=0

Figura 4: Protocolo capa de transporte TCP

Cuando se envían mensajes del tipo texto, se puede observar paquetes usando el protocolo TCP (como se muestra en la figura 5). A diferencia del caso anterior, esta vez se usó una localidad diferente para realizar la prueba, por lo que en esta ocasión la dirección correspondiente a nuetro computador personal es 10.2.43.251, y la otra corresponde al servidor receptor. Otro detalle observable es que todos los paquetes van dirigidos al puerto 443 del servidor receptor, como se había explicado anteriormente, confirmando el funcionamiento del servicio de mensajería de texto. En su mayoría se presentan paquetes del tipo ACK dado que la conexión se mantiene activa, sin necesidad de volver a enviar un requerimiento.

	Time	Source	Destination	Protocol	Length	Info
26	0.480631	192.168.1.101	185.50.107.71	TLSv1.2	251	Client Hello
29	0.551723	185.50.107.71	192.168.1.101	TLSv1.2	1514	Server Hello
32	0.553460	185.50.107.71	192.168.1.101	TLSv1.2	420	Certificate, Server Key Exchange, Server Hello Done
34	0.555001	192.168.1.101	185.50.107.71	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
36	0.626525	185.50.107.71	192.168.1.101	TLSv1.2	312	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
37	0.640228	192.168.1.101	185.50.107.71	TLSv1.2	665	Application Data
38	0.709777	185.50.107.71	192.168.1.101	TLSv1.2	265	Application Data
39	0.709778	185.50.107.71	192.168.1.101	TLSv1.2	134	Application Data
41	0.712592	192.168.1.101	185.50.107.71	TLSv1.2	257	Application Data
43	0.785145	185.50.107.71	192.168.1.101	TLSv1.2	226	Application Data
52	1.751880	192.168.1.101	185.50.107.71	TLSv1.2	674	Application Data
53	1.758278	192.168.1.101	185.50.107.71	TLSv1.2	135	Application Data
54	1.758375	192.168.1.101	185.50.107.71	TLSv1.2	147	Application Data
58	1.817829	185.50.107.71	192.168.1.101	TLSv1.2	359	Application Data
69	2.867605	192.168.1.101	185.50.107.71	TLSv1.2	135	Application Data
98	3.280694	192.168.1.101	185.50.107.71	TLSv1.2	91	Application Data
99	3.286990	185.50.107.71	192.168.1.101	TLSv1.2	100	Application Data
100	3.287698	185.50.107.71	192.168.1.101	TLSv1.2	85	Encrypted Alert

Figura 5: Protocolo de seguridad TSL v1.2

Finalmente, en cada una de las conexiones anteriores, se tiene que el sistema de protección y seguridad hace efecto en cada caso, abriendo con los mensajes "Hello"del cliente y servidor, para

luego comenzar con el intercambio de paquetes que contienen la información para la conexión a servidor.

5. Conclusión

En primer lugar, la separación entre los procesos de comunicación vía voz y texto permiten la regularización de ambos de manera particular, permitiendo además aplicar a cada uno los protocolos más adecuados para su correcta implementación. Con esto nos referimos a que para el caso de comunicación por voz, dado que estamos hablando de aplicación a juegos en línea, se necesita que sea lo más rápida posible, pudiendo tomar el riesgo de que ciertos paquetes puedan perderse ya que dado como está estructurada nuestra lengua, lo ideal sería obtener el mensaje completo, pero no es estrictamente necesario, por lo que se utilizará un protocolo UDP para la capa de transporte. En el caso de la comunicación por texto, esta es necesaria que llegue completa y de forma segura, por lo que el protocolo adecuado para la capa de transporte sería el TCP, dada su alta fiabilidad.

El uso de Websockets permite una conexión segura a través de un canal fijo a partir del protocolo TCP, sin necesidad del uso de paquetes de confirmación para cada request del cliente, esto significa una amplia mejora en el tiempo de comunicación, y dado que estamos hablando de comunicación en tiempo real, cada ahorro que se pueda obtener en términos de tiempo es valioso, sobre todo por el uso del sistema de codificación JSON utilizado, ya que necesita compresión y descompresión, procesos que toman tiempo en resolverse.

6. Referencias

1. https://discordapp.com/developers/docs/topics/
2. https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/
3. https://www.swhosting.com/blog/transport-layer-security-tls-que-es-y-como-funciona/
4. https://www.toptal.com/chatbot/how-to-make-a-discord-bot
5. https://www.baeldung.com/rest-vs-websockets