Comunicaciones Cliente-Servidor de baja latencia ambientado a Videojuegos.

Integrantes:

Gabriel Carrillo

Daniel Espinoza

Rodrigo Gallardo

Resumen:

Al programar las comunicaciones entre aplicaciones, es muy importante elegir los protocolos y estrategias de comunicación adecuados para la tarea que se desea realizar. Distintos protocolos de comunicación tendrán distintas ventajas y desventajas, y es necesario tener esto en cuenta para escoger la mejor solución posible.

La comunicación para videojuegos online, por ejemplo, a menudo debe ser liviana y con poca latencia, y en caso de pérdida, no se debe esperar a estos paquetes, sino que descartarlos para evitar latencia. Esto hace que cualquier protocolo que sea “confiable”, es decir, que se encargue de la espera o recuperación de paquetes, sea inadecuado para esta aplicación.

Introducción:

En el comienzo del aprendizaje sobre la complejidad de las redes de computadores, naturalmente se menciona el uso de una técnica indispensable para estudiar este tema, una técnica que va a dividir nuestro problema de análisis en varios subproblemas de igual tamaño y la solución global, será juntar todas las soluciones de los problemas más pequeños en comparación al original. Ésta técnica ha sido la que hemos empleado para el enfoque de las redes de computadores y en este informe, analizaremos en detalle la relación existente entre factores de uno de esos subproblemas, específicamente, de la Capa de Transporte.

Tenemos entonces que la Capa de Transporte tiene como función fundamental la transferencia de datos entre transmisor y receptor, aunque no estén conectados directamente. Por ende, existen protocolos que definirán sus propias políticas en este contexto, considerando una transferencia fiable de datos, teniendo en cuenta la congestión del entorno o simplemente no preocuparse por si llegarán bien los datos a destino, por decir algunas.

El mayor inconveniente del uso de estas políticas es decidir cuál es mejor para qué contexto y por qué. Si ejemplificamos alguna situación en la práctica, las páginas web requieren un establecimiento de conexión previo para ser ingresadas y poder enviar solicitudes al servidor, generando mayor seguridad y control al movimiento de información, y así funcionaría también enviar un correo a un cierto destinatario. Pero ¿qué sucede precisamente con otros contextos como videojuegos en línea, donde toda acción debe suceder lo más pronto posible? Para ello, se mantiene la premisa de sacrificar confiabilidad y seguridad, para mayor velocidad.

El código utilizado para implementar la comunicación entre aplicaciones es comúnmente llamado “netcode”.

Implementación de “Netcode” en Videojuegos Online:

El netcode de una aplicación se construye sobre los protocolos de transferencia

Estrategias:

- Minimizar el tamaño de los paquetes:
  Cuando se busca maximizar la velocidad de la comunicación por redes, es importante considerar el tamaño de los paquetes que se mandan y la congestión que estos conllevan. Es por esto que se busca minimizar la cantidad de datos que se enviarán por cada paquete.
- Minimizar la cantidad de paquetes enviados por segundo:
  Al igual que el punto anterior, es importante considerar la cantidad de paquetes que se enviarán cada segundo. Idealmente, un jugador podría ver su posición y estado actualizado en cada cuadro mostrado por su monitor. En general, el número de cuadros por segundo que mostrará un monitor es 60, es decir, el usuario tendría que recibir y procesar 60 paquetes de estado cada segundo. Algunos juegos, como Counter-Strike:Global Offensive, implementan comunicaciones a esta velocidad, con 128 “ticks” por segundo siendo el estándar competitivo, pero para la mayoría de los juegos no serán necesarias comunicaciones con tan poca latencia.
- Interpolación de estado y simulaciones paralelas:
  Con el objetivo de minimizar la cantidad de paquetes enviado al cliente sin que los movimientos se vean entrecortados, es necesario producir cuadros intermedios entre cada actualización. Hay dos formas comunes de implementar esto; una de ellas es la interpolación de estados, en la cual se mantiene un buffer con los dos últimos estados recibidos y se hace una simple interpolación entre ellos. La otra es que tanto el cliente como el servidor lleven simulaciones paralelas, en cual caso el servidor debe proporcionar no sólo la posición de los objetos en el juego, sino también los vectores de velocidad y aceleración pertinentes.

Para comenzar el proceso de investigación e implementación de lo que queremos mostrar, como objetivo de hacer funcionar una conexión cliente-servidor usando un socket con cierto protocolo de transporte ambientado a los videojuegos en línea, presentamos las siguientes herramientas:
- Programación en C++ junto a una librería llamada Qt para manipular sus respectivas funciones.

Cuando se habla sobre la conexión en videojuegos en línea, los datos que son enviados al respectivo servidor son generalmente las acciones que queremos

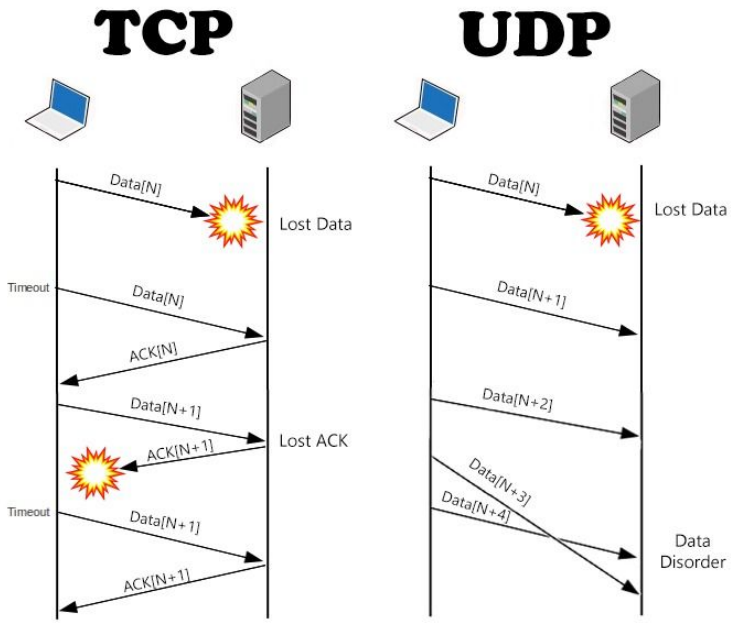
realizar en ese momento, ya sea presionando una tecla de un computador y de un controlador externo que realice unas funciones equivalentes a lo que queremos lograr, recibiendo como respuesta del servidor la acción del personaje. Para ello, la librería que se estará utilizando contendrá funciones tales que implementen sockets UDP, además de relacionar las teclas presionadas y soltadas para que el servidor comprenda la acción que queremos realizar. A continuación, comenzaremos con un análisis comparativo entre los dos protocolos de la Capa de Transporte, para así vislumbrar la importancia de cada uno de acuerdo a un determinado contexto.

Comparación de protocolos de transmisión de datos:

Protocolo TCP	**Protocolo UDP**
Es un servicio orientado a la conexión, que realiza un proceso llamado “handshaking”, previo a la comunicación entre transmisor y receptor.	Es un servicio que no está orientado a la conexión. Por ende, la transmisión de datos comienza inmediatamente.
Garantiza políticas amigables en cuanto a la transmisión de datos se refiere, como seguridad, control de congestión, control de flujo, recepción ordenada de datos, entre otros.	Este protocolo no garantiza que los datos serán correctamente enviados a destino, ni tampoco posee políticas favorables para cada transmisión.
Posee altos requisitos de carga y además, permite exclusivamente la comunicación punto a punto.	Posee bajos requisitos de carga y además, permite la comunicación punto a punto y de un punto a varios.
Este protocolo es muy útil para aplicaciones que requieren una confiabilidad muy alta. Por ende, el tiempo de transmisión es menos crítico.	Este protocolo funciona muy bien para aplicaciones que requieren una transmisión rápida y efectiva de datos. Por ende, es muy utilizado por servidores que reciben una gran cantidad de solicitudes pequeñas de un alto número de clientes.
Es un sistema relativamente lento para el transporte de datos, debido a todos los procesos previos a la transmisión, y a la alta carga que requiere su implementación.	Es un sistema bastante ágil para el transporte de datos, debido a que su prioridad fundamental es la transmisión instantánea.
HTTP, HTTPs, SMTP y Telnet, por decir algunos, son protocolos que utilizan TCP.	DNS, DHCP, TFTP, SNMP, RIP y VoIP, por decir algunos, son

	protocolos que utilizan UDP.

El comportamiento de los protocolos mencionados está ejemplificado por el siguiente diagrama:



Donde TCP, debido a que se se encarga de que no ocurra ni pérdida de paquetes ni llegada de paquetes fuera de orden, introduce latencia extra llevando a cabo funciones que muchas veces son innecesarias. Por ejemplo, si los paquetes enviados por el servidor corresponden a la posición del jugador en un determinado instante, no es necesario que el cliente reciba los paquetes más antiguos, ya que la posición más nueva toma preferencia y reemplaza la posición antigua. Esto nos permite descartar los paquetes más antiguos sin que se pierda información importante para el jugador.

En juegos donde la latencia de la comunicación no es de mayor importancia, a menudo se usa TCP para asegurar la llegada de los paquetes al cliente. Esto es posible en juegos de los géneros RTS(Real Time Strategy), MMORPG(Massively Multiplayer Online Role Playing Game), y cualquier tipo de juego por turnos, por nombrar algunos.[ref002]

Protocolos Híbridos:

Existen también implementaciones de netcode que mezclan estos dos protocolos, donde distintos tipos de paquetes utilizan el protocolo más adecuado para el objetivo que deben cumplir. Esto le permite a los desarrolladores aprovechar las ventajas de ambos protocolos sin tener que diseñar una solución muy compleja, pero también puede traer problemas, ya que el estudio de estos protocolos ha demostrado experimentalmente que el tráfico TCP en una red afecta negativamente la eficiencia de los sockets Udp en esta misma, causando pérdidas[ref001].

Implementaciones especiales de TCP:

En el caso de algunas aplicaciones, que desean mantener las ventajas de TCP, pero optimizar su comportamiento para que sea más apropiado al contexto, también existen implementaciones especiales de TCP, que definen comportamientos diferentes al encontrar pérdida de paquetes o paquetes fuera de orden.[ref002]

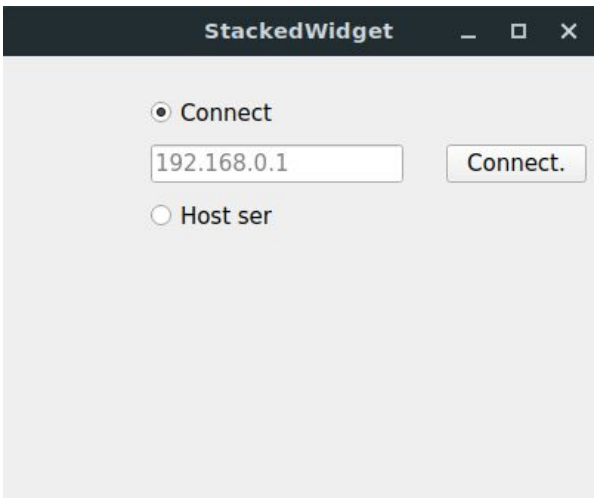
Nuestra Implementación:

Usando Qt y c++, programamos una aplicación gráfica que sería capaz tanto de establecer un servidor vía sockets UDP como conectarse a un host ya establecido.

Una vez que se elige el modo de ejecución, se crean los objetos necesarios para mostrar las gráficas de la aplicación(QGraphicsView, QGraphicsScene, y los objetos correspondientes a los jugadores, los cuales heredan desde QGraphicsItem para hacerlos dibujables), además de un socket para comunicar el cliente con el servidor. Adicionalmente, en el caso de que se elija ser el host, se crea un objeto de la clase UdpServer, definida por nosotros y que se encarga de recibir las acciones de los jugadores y actualizar sus posiciones.

Debido a problemas internos al equipo las metas de nuestra implementación no pudieron ser alcanzadas, y problemas en el diseño de nuestra solución no pudieron ser corregidos a tiempo. El servidor es capaz de registrar nuevos jugadores a partir de paquetes recibidos desde direcciones desconocidas y de identificar aquellas direcciones a las que ya se les ha asignado un jugador, pero surgieron problemas al implementar las clases nativas a Qt que de momento impiden la lectura correcta del bloque de datos en un datagrama UDP.

Capturas de pantalla del diálogo mostrado al iniciar el programa y de la venta abierta al conectarse a un servidor.



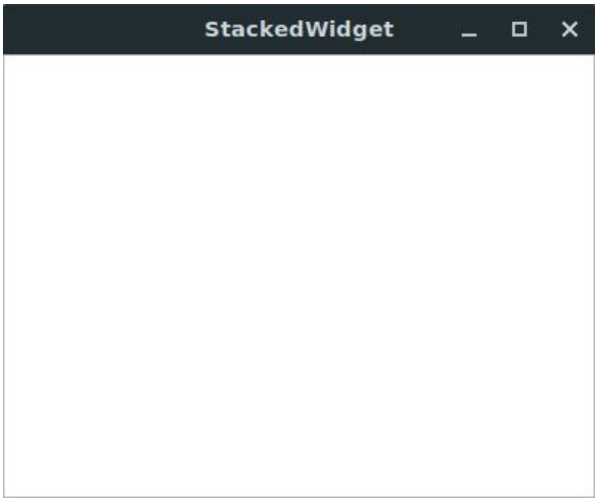
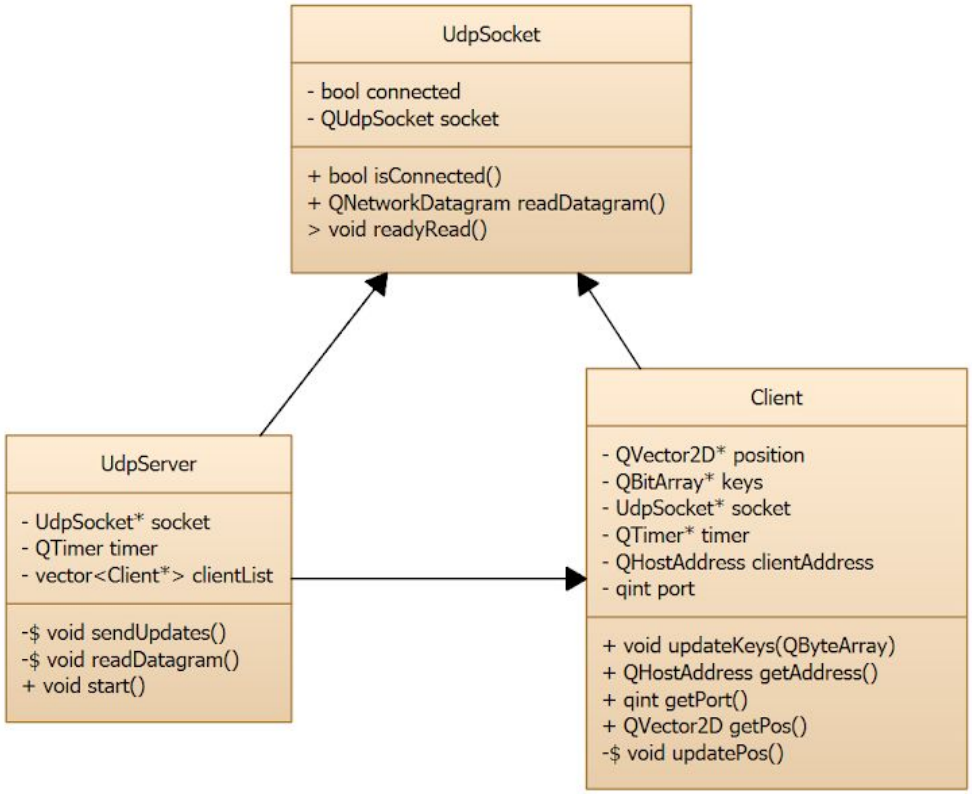


Diagrama simple de las clases usadas para establecer el servidor UDP.



Conclusiones:

Podemos concluir que, considerando todos los factores desarrollados para incorporar un enfoque más específico a la hora de comprender la transmisión de información de la forma cliente-servidor en un contexto determinado, en redes de computadores, la construcción de soluciones para abordar problemas que consuman cantidades de espacio, tiempo o ambas, dependiendo del contexto, se velará por la mejor forma de resolverlos, aun así si se considera descartar el uso de políticas de seguridad en relación a la transmisión de los datos. Además, hemos mostrado que el funcionamiento que debe realizar la Capa de Transporte dependerá de cierto contexto y por ende, tenemos que ser críticos ante cualquier implementación práctica que se desarrolle al respecto.

Finalmente, el modo de operación que utilizamos para abordar el problema, no es un enfoque que se aplique exclusivamente a las redes de computadoras, es una forma de preferir lo mejor de acuerdo a un contexto determinado, ya que el protocolo TCP, en el manejo de videojuegos que requieren bajas latencias en línea, no será la opción más óptima, ya que conocemos la razón de por qué no funciona correctamente, y es que el algoritmo de control de congestión que posee este protocolo, afecta a la velocidad que como desarrolladores esperamos alcanzar, y por esta razón, preferimos UDP en este caso. Podemos relacionar además, que en diversos contextos prácticos se va a criticar lo mismo, sobre el qué queremos, y qué debemos sacrificar para lograrlo.

Referencias:

[ref001] Characteristics of UDP Packet Loss: Effect of TCP Traffic

[ref002] Game Traffic Analysis: An MMORPG Perspective