Comunicaciones punto a punto

Introducción

El nivel de enlace es el encargado de realizar una conexión a nivel local. Aunque el nivel físico define la forma de enviar bits, el nivel de enlace forma mensajes con esos bits, capaces de no solo llevar los datos del usuario, sino de controlar el buen funcionamiento de la conexión.

Las tres funciones principales del nivel de enlace consisten en el framing, el control de flujo y la corrección de errores, siendo el primero de estos tres vital para garantizar el correcto funcionamiento de un enlace. Como objetivo secundario, el nivel de enlace puede definir formas de usar más eficientemente el canal, usualmente asociadas al control de errores.

Framing

Framing es el acto de delimitar el comienzo y fin de un mensaje. Esto se denomina framing, porque este acto crea un marco (frame) en donde el mensaje puede ser enviado. Por esto, en general a los paquetes de nivel 2 se los denomina “frames”. El Framing es uno de los conceptos vitales del nivel de enlace, sin el cual no se tiene una capa que funcione efectivamente.

Para el framing o entramado existen varias estrategias con diferentes atributos. Estas estrategias se pueden agrupar en estrategias de stuffing, estrategias con campos de longitud, violación de código y de longitud fija.

Entre las estrategias de stuffing, las dos principales son bit stuffing y byte stuffing. Bit stuffing consiste en definir un delimitador (usualmente 01111110) y enviar los datos precedidos y antecedidos por el delimitador. Si fuera necesario mandar datos que serían idénticos al delimitador, entonces debe haber una regla que haga que este dato no sea escrito literalmente, pero que al mismo tiempo se entienda lo que se deseaba enviar. Un ejemplo de esta regla, que usa el delimitador mencionado anteriormente, consiste en insertar un cero incondicionalmente cada vez que se envían cinco unos. Si se deseara enviar el delimitador como datos, lo que lógicamente se escribiría como 01111110, físicamente se escribirá como 011111010, evitando así insertar un delimitador que cortaría prematuramente el paquete. Es importante notar que esta regla debe aplicarse aún cuando lo que se desee enviar despues de los cinco unos sea un cero. Puesto que el receptor no conoce lo que el emisor desea enviar, removerá incondicionalmente el cero siguiente a los cinco unos.

Byte stuffing es una estrategia similar a nivel de byte. Esta estrategia también se parece a la forma de indicar caracteres de control en las cadenas de varios lenguajes de programación. En el caso de byte stuffing, se define un delimitador y un caracter de escape. El delimitador indica el principio y fin del frame. Si se deseara enviar el delimitador como dato, este es precedido por el caracter de escape (lo que indica que no es el verdadero delimitador). Si lo que se desea enviar es el caracter de escape, se lo envía dos veces (o sea se lo envía escapado) para que el receptor sepa que se lo debe tomar como dato literal. Esta estrategia es similar a la usada en los strings de C para el tratamiento de caracteres de control.

Otra forma de indicar el fin de un frame consiste en indicar la longitud de un mensaje en un campo del encabezado. Esta longitud debe encontrarse en un lugar predecible, ya que será necesario interpretarla correctamente para determinar donde termina un frame. En general la longitud se encuentra en el encabezado, proxima al comienzo y antes de cualquier campo variable.

Otra estrategia para indicar el final de un frame se llama violación de código. Esta estrategia se apoya en el nivel inferior, basándose en el hecho de que pueden haber señales que no se correspondan con cero y uno (por ejemplo una señal que indique ausencia de datos). Para delimitar un frame, se deja un espacio en el cual no se señaliza ni cero ni uno, lo cual es detectado por el

receptor, que lo interpreta como un espacio entre frames.

Una última estrategia consiste simplemente en tener frames de longitud fija. Para ello, el emisor puede indicar de alguna forma que está enviando un frame y el receptor simplemente recibe una cantidad de bytes predefinidos o puede tratarse de una estrategia en la cual se deba transmitir constantemente y un reloj marque el principio y fin de cada frame.

El resultado final en cualquiera de estos casos es la identificación de donde empieza y termina cada frame.

Control de errores

El control de errores consiste en asegurar que los mensajes lleguen, lleguen bien y lleguen en orden. En el nivel de enlace, que los mensajes lleguen en orden no es un problema, ya que el medio no desordena. El foco principal en este nivel es que los mensajes lleguen correctamente (y que no se pierdan o lleguen con errores).

Distancia de hamming

Antes de poder hablar de códigos de detección y corrección de errores, es necesario conocer algunas propiedades básicas de codificación en general.

Una propiedad importante es la distancia de hamming. Entre dos codificaciones (ambas de igual longitud), la distancia de Hamming es la cantidad de símbolos que hay que cambiar para que se pueda pasar de uno a otro. Aplicado a un código entero, es la menor distancia de hamming entre todas las cadenas que pueden formarse con el mismo.

Esencialmente, la distancia de hamming de un código es la mínima cantidad de modificaciones a realizar en un mensaje válido para que el resultado sea otro mensaje válido. Un ejemplo es un código de longitud fija (3 bits por símbolo) en donde solamente los códigos “000” y “111” son válidos. En este caso, la única forma de pasar de un código válido a otro es convertir los tres símbolos. Este código tiene una distancia de hamming de tres.

Métodos para detectar y corregir errores

La distancia de hamming es importante ya que determina si un error puede ser detectado o si un mensaje puede ser corregido, dadas las circunstancias que causaron el error.

Si un código tiene una distancia de hamming d, y al transmitir un símbolo se producen a lo sumo n errores, donde N se puede expresar como

$D = N + 1$

entonces es posible detectar la ocurrencia del error.

Adicionalmente, si ocurrieron a lo sumo M errores, tal que se cumpla

$D = 2 * M + 1$

entonces aparte de poder detectar el error, va a ser posible corregirlo.

Para el ejemplo de arriba, podemos ver que D = 3. Entonces eso significa que si durante la transmisión de un símbolo ocurren a lo sumo dos errores, será posible detectarlo, ya que no hubieron suficientes cambios para pasar de “000” a “111” o viceversa.

Usando la segunda fórmula, podemos saber que si en el envío de un mensaje ocurre un error (y no más de un error), será posible corregirlo. Esto se puede ver facilmente si se evaluan las distancias de hamming a cada posible valor. Asumiendo que se recibió “010”, podemos medir la distancia de hamming a cada uno de los posibles valores (obteniendo una distancia de 1 para “000” y de 2 para

“111”). Sabiendo que la distancia máxima recorrida es 1 (por la cantidad de errores máximos), el valor enviado debió haber sido “000”. La corrección esencialmente consiste en movernos al valor válido más cercano.

Si no se respetan estas condiciones, entonces el resultado de una detección o corrección puede ser equivocado. Si Se producen suficientes errores, se podrá pasar de un código válido a otro, con lo que no se detectará un error que en realidad sucedió. Si no se cumple la limitación para la corrección, es posible que un intento de corrección lleve al símbolo equivocado (ya que hubieron suficientes errores para que lo recibido se acercara demasiado al símbolo equivocado) o simplemente que falle el intento de corrección.

Métodos usados en la práctica

En la práctica, los métodos de detección y corrección de errores pueden variar desde métodos extremadamente simples, que se modelan facilmente con la distancia de hamming a métodos más complejos que son bastante eficaces a pesar de que al ser analizados desde el punto de vista de la distancia de hamming parecen ser algoritmos pobres.

Uno de los algoritmos más simples es el bit de paridad. Esta técnica consiste en agregar un bit en alguna parte de los datos (usualmente al final) y hacer que el valor de este nuevo bit dependa de los valores de los bits del dato (por ejemplo, hacer que la cantidad de unos sea par en el mensaje resultante). Analizándolo con las técnicas del punto anterior se puede ver que su distancia de hamming es 2. Esto nos dice que puede detectar errores de un bit y que no puede corregir. Efectivamente, el bit de paridad puede detectar una cantidad impar de errores.

Una variante de la paridad consiste en ordenar los datos a enviar en un cuadrado y calcular las paridades por filas y columnas. Este método tiene una distancia de hamming de tres, por lo que puede corregir errores de un bit y detectar errores de hasta dos. Esto se debe a que un error que ocurra en los datos será indicado por dos de los bits de paridad, los que marcan en que fila y columna ocurrió el error. Para que un error no sea detectado, el cambio mínimo que debe lograrse es alterar un bit y sus dos bits de paridad. Una alteración menor a esto será detectada.

Un algoritmo muy usado es el CRC. Este algoritmo se basa en considerar al mensaje como un polinomio y luego dividirlo por un polinomio divisor definido de antemano. Una vez hecha la división, el resto se manda con el mensaje. El receptor puede realizar la misma división y verificar que el resto que obtuvo coincida con el enviado. Si se analiza este algoritmo viendo la distancia de hamming, se notará que tiene una distancia de dos, lo que es bastante pobre. Sin embargo, el CRC posee varias propiedades no contempladas en este análisis: es facil y rápido de calcular usando operaciones de bits, puede detectar cualquier error que altere una cantidad impar de bits y siempre va a detectar un error de ráfaga que no exceda el tamaño del polinomio divisor. Estas propiedades lo hacen deseable en la práctica, a pesar de que dos alteraciones puestas en los lugares correctos puedan hacer que un error no sea detectado.

Entre los códigos de corrección de errores usados se destaca Reed-Solomon, que se basa en interpretar el mensaje como una función polinómica, de la cual se toman más muestras que las necesarias para reconstruirla. Esto permite reconstruir el mensaje aunque falten algunos datos, o hayan datos erróneos.

La desventaja de la corrección de errores sobre la detección es que los códigos de corrección son mucho más largos que los de detección, lo que agrega un overhead a la comunicación. Salvo en casos muy especiales en donde la tasa de errores sea alta o el costo de restransmisión sea grande, en general es común encontrar algoritmos de detección combinados con un esquema de retransmisión de mensajes.

Métodos de entrega confiable

Para que un protocolo sea confiable, es necesario tener un mecanismo que permita la retransmisión en caso de errores. Para ello, al menos es necesario numerar los mensajes y tener alguna forma de indicar que un mensaje dado llegó a destino.

Usualmente la numeración consiste en un campo de tamaño limitado que contiene un número de secuencia. Cuando el número de secuencia llega a su valor más alto, se lo continúa con el valor cero, reiniciando el ciclo. Para indicar que un frame llegó a destino, se usa un mensaje de control llamado ACK, que indica que hasta cierto punto la transmisión tuvo éxito.

Una técnica básica es "stop & wait", que se basa en mandar un frame y esperar su asentimiento. Puesto que los mensajes ACK pueden perderse, es necesario al menos usar dos números de secuencia para los mensajes. Si un mensaje o su ACK se pierden, se espera un tiempo y al no ver el ACK correspondiente se declara un timeout y el mensaje se retransmite.

Stop & wait es una técnica ineficiente, ya que es necesario esperar a que un mensaje sea asentido antes de poder enviar uno nuevo. Si el RTT del medio es alto, o el medio tiene mucha capacidad, gran parte del tiempo permanecerá ocioso por falta de datos para enviar.

Para solucionar este problema, se pueden enviar varios frames uno atrás de otro sin esperar el asentimiento. Esta es la base para la técnica de "go-back-n". En go-back-n, se debe estimar cuanta información puede estar en vuelo en el medio durante un RTT. En base a esta cantidad se asignan buffers en el emisor, de forma tal que en ellos se pueda guardar al menos la cantidad de información necesaria para usar el medio durante un RTT.

Cuando hay frames para enviar, estos se almacenan en los buffers, se les asigna un número de secuencia y se los despacha por el medio, uno tras otro. La cantidad de buffers define una "ventana", que representa a los datos que son transmitidos pero que no necesariamente están asentidos. Puesto que el tamaño de esa ventana es proporcional al RTT, se espera que se puedan mandar frames constantemente si esperar el asentimiento, lo que mantiene el canal lleno. Cuando esta ventana se esté acabando, la llegada de los ACK de los primeros frames permiten al emisor enviar nuevos frames, estos ACK "corren" la ventana y permiten el envio de nuevos datos.

En el receptor solamente hace falta un buffer. Si se perdiera algún frame, debido a que la única forma de retransmitir es llegar a un timeout se deberá volver al punto en donde ocurrió la pérdida y reiniciar la transmisión desde ahí. Agregar buffers en el receptor no ayuda, ya que es de esperar que los timeouts de los paquetes siguientes se venzan apenas la retransmisión salga por el medio.

Un detalle que surge acá es definir el tamaño de los valores usados para secuenciar los mensajes. Si el tamaño es demasiado pequeño (por ejemplo, suponiendo que pueden haber 4 frames en vuelo y los valores de número de secuencia tienen dos bits), puede ocurrir que los frames de una ventana lleguen a destino, pero que sus ACK se pierdan. En este caso, el emisor retransmitirá los datos, pero el receptor los aceptará como nuevos, lo que causa que se dupliquen los datos a subir al nivel 3.

Para evitar esto, debe cumplirse la siguiente inecuación:

$\#S >= Vt + Vr$

En donde #S es la cantidad de valores distintos usados en los números de secuencia, Vt es el tamaño de la ventana de transmisor (en frames) y Vr es el tamaño de la ventana de recepción (o sea los buffers que posee el receptor para almacenar mensajes, en frames).

Esta fórmula, aplicada a go-back-n, indica que $\#S = Vt + 1$, o sea que se debe usar al menos un valor adicional a la cantidad de frames que pueden estar en vuelo en un momento dado. Esta fórmula también dice que si se agranda la ventana del receptor (aunque no mejore la performance) se deben agregar más valores de secuencia, o se corre el riesgo de introducir errores.

Una técnica más tolerante a errores consiste en agregar un segundo mensaje, NAK, el cual indica

que hubo una pérdida y que el frame indicado debería ser reenviado a la brevedad. Adicionalmente se deben agregar buffers en el receptor, lo que incrementa la ventana de recepción, y se deben agregar números de secuencia adicionales para cubrir este incremento. Esta técnica es conocida como “rechazo selectivo”.

Mientras no haya pérdida de datos, el algoritmo funciona de forma similar a go-back-n. Al ocurrir una pérdida, el receptor se puede dar cuenta porque llega un mensaje posterior al esperado. Puesto que el receptor tiene buffers adicionales, puede guardar este mensaje posterior y enviar un NAK pidiendo el mensaje faltante. Este mensaje posterior se encuentra “dentro de la ventana de recepción”. El receptor no puede mandar un ACK, ya que esto correría la ventana, pero puede prepararse para cuando el emisor mande el frame faltante almacenando los frames que recibe en sus buffers.

Cuando el emisor ve un NAK, retransmite inmediatamente el frame faltante sin esperar al timeout. El objetivo de esto es cubrir el agujero en los datos que tiene el receptor antes del vencimiento de los otros frames. Si la retransmisión tuvo éxito, el receptor puede cubrir ese faltante y luego anuncia un ACK que desplaza la ventana hasta el último mensaje que tenía almacenado, o hasta el siguiente agujero, si lo hubiera. Gracias a esto, el emisor evita retransmitir una serie larga de mensajes y puede usar el medio para enviar otros frames.

En rechazo selectivo, la cantidad de buffers del receptor es idéntica a la del emisor. Esto le permite al receptor almacenar todos los datos que pudiera enviar el emisor y al mismo tiempo poder optimizar la respuesta cuando los agujeros en los datos sean cubiertos. Tener más ventanas no ayuda al receptor, ya que el emisor no las usa y tener menos puede significar que el receptor deba descartar mensajes del emisor por falta de espacio.

Control de flujo

El control de flujo consiste en evitar que un emisor rápido y con muchos recursos inunde a un receptor lento. Para ello, un receptor que está siendo inundado debe poder avisar al emisor que debe parar de enviar datos. Esto se puede señalizar de forma explícita o implícita.

En la señalización explícita, se definen dos mensajes de control, normalmente conocidos como RR (receiver ready) y RNR (receiver not ready). Cuando el receptor está siendo inundado, envía un RNR al emisor. El emisor al ver este mensaje para el envío. Cuando el receptor puede volver a recibir, envía un RR, lo que reinicia el envío de datos.

Esto se puede hacer de forma implícita, mintiendo en el control de errores. Cuando un receptor está siendo inundado, puede dejar de asentir los frames que van llegando. El emisor asume que no están llegando los frames y no avanza la ventana, lo que permite que estos frames sean recibidos luego, cuando el receptor tenga recursos para hacerlo.

Es importante aclarar que el control de flujo es mucho más simple en el nivel de enlace que en otros niveles, ya que en este caso los tiempos de reacción que tiene el emisor son conocidos, ya que las distancias y tiempos de propagación son acotados. Esto no siempre es así en otros niveles.

También es importante notar que la red tiene capacidad suficiente para el envío, y que la falta de recursos se encuentra en el receptor. No hay que confundir el control de flujos con el control de congestión que se verá más adelante, en donde el cuello de botella está en la red.