DISEÑO Y PUESTA EN MARCHA DE UN SISTEMA EMBEBIDO PARA LA COMUNICACIÓN BIDIRECCIONAL ENTRE DOMINIOS CON DIFERENTES RESTRICCIONES TEMPORALES

CHRISTIAN L. GALASSO

Ingeniero en Electrónica (UTN-FRBB), docente del Posgrado en Análisis Operativo (ESOA). Investigador de la UNDEF, cat. C; director del grupo de I+D Soluciones Embebidas Aplicadas – UNDEF – Facultad de la Armada – ESOA.

MIGUEL ÁNGEL BANCHIERI

Ingeniero en Electrónica (UTN-FRBA), magíster en Ingeniería Biomédica (Universidad Favaloro) y profesor titular ordinario con dedicación exclusiva (UTN-FRBB).

FRANCO CASPE

Ingeniero en Electrónica (UTN-FRBB), becario en PIDDEF 08/12. En 2017, trabajó como ingeniero en software para Hellastorm. En 2018, fue pasante IAESTE en Coimbatore, (India) en diseño de antenas.

MARTÍN E. PAZ

Estudiante de la carrera de Ingeniería Electrónica de la UTN FRBB. Técnico Universitario en Electrónica 2018. Integrante del Grupo de Robótica y Simulación de la UTN FRBB. Investigador del grupo de I+D Soluciones Embebidas Aplicadas. UNDEF – Facultad de la Armada – ESOA.

Resumen

A continuación, se describe el diseño de un protocolo de comunicaciones que permite el intercambio de información entre una red de ordenadores de aplicación específica, que opera en tiempo real, y una PC comercial que

ejecuta una aplicación sobre un sistema operativo de tiempo diferido. Se logra una comunicación transparente entre ambos extremos implementando un *gateway* (o puerta de enlace) compatible, en un sistema electrónico embebido, desarrollado anteriormente por el grupo de investigación, que oficiará también como un "latch", el cual porta información entre dominios con reglas de operación incompatibles entre sí.

Palabras clave

Protocolo, Puerta de Enlace, Tiempo Real, Control Remoto, Microcontrolador, Sistema Embebido, Interoperabilidad.

Abstract

The following paper describes the design of a communications protocol that allows the exchange of information between a network of computers with a specific application, which operates in real time, and a commercial PC that executes an application on a deferred time operating system. Transparent communication between both ends is achieved by implementing a compatible *gateway* (or *gateway*), in an embedded electronic system, developed previously by the research group, which will also act as a "latch", which carries information between domains with rules of operation incompatible with each other.

Keywords

Protocol, Gateway, Real time, Remote control, Microcontroller, Embedded System, Interoperability.

Introducción

Desde hace más de tres décadas existen en servicio un sinnúmero de líneas de control y accionamientos remotos en una variedad de campos de interés tales como la generación eléctrica, la navegación y la aeronáutica. La revolución tecnológica ha transformado profundamente la interrelación entre los sistemas digitales, logrando relevamientos más sencillos de las condiciones de funcionamiento, garantizando operaciones cada vez más

seguras y amenizando las interfaces humanas. A la luz de estos hechos, es evidente que equipamiento específico, que está operativo y cuyo reemplazo completo es en exceso oneroso, podría encontrar una alternativa de modernización gradual (económicamente más viable) si pudiera desarrollarse una suerte de *gateway* que implemente un protocolo que permita vincular tecnologías obsoletas con actuales.

Considerando entonces los casos de interés mencionados, existen numerosos ejemplos de aplicación de sistemas informáticos antiguos, diseñados ad hoc, en que la vida útil de la planta, por su diseño, queda fuertemente vinculada a la vida útil del dispositivo que la controla.[1] La posibilidad, a futuro, de operar estos sistemas depende entonces de la capacidad de adaptarlos, como parte de una estrategia de reingeniería,[2] a los nuevos conceptos del estado del arte, los cuales exigen interfaces que puedan interaccionar a distancia, y con compatibilidad multiplataforma.[3]

Para el caso específico de este trabajo, se tiene un sistema antiguo de tiempo real duro,[4] de arquitectura cerrada, cuyas unidades funcionales están distribuidas en distintos ordenadores en red, interconectados entre sí por varias placas de comunicaciones de tecnología propietaria, formando una topología tipo malla, aunque con un nodo central bien diferenciado. Interesa entonces generar un enlace full dúplex entre este sistema cerrado y un dispositivo externo, como puede ser una PC comercial, con los objetivos de monitorear de forma externa las variables de estado que se controlan, y también poder realizar pruebas y desarrollar simulaciones, al contarse con

---

1 Ejemplo de un sistema informático de tiempo real que opera sin modificaciones durante todo el ciclo de vida de una planta. Nuclear plant powers up on real-time OS. Disponible en: http://www.itbusiness.ca/news/nuclear-plant-powers-up-on-real-time-os/9084.

2 Manejo de la obsolescencia tecnológica: Suresh K. Nair. A model for equipment replacement due to technological obsolescence. *European Journal of Operational Research* 63 (1992) 207-221 Disponible en: https://www.researchgate.net/profile/Wallace_Hopp/publication/4941721_A_Model_for_Equipment_Replacement_Due_to_Technological_Obsolescence/links/57f3e72e08ae886b897dccad.pdf.

3 Ejemplo de pérdida de funcionalidad y adaptabilidad de una planta debido a la antigüedad de los sistemas informáticos vinculados. Ver: Re-programming "Little Boy". Adelanto sobre la reestructuración de Ferrania. Disponible en: http://www.filmferrania.it/news-articles/2017/welcome-to-2017.

4 Categories of real time systems. Giorgio C. Buttazzo. *Hard Real-Time Computing Systems.* Disponible para vista previa en: https://books.google.com.ar/books?id=h6q-e4Q_rzgC&printsec=frontcover&hl=es.

la posibilidad, tanto de interpretar como de generar los mensajes de varios periféricos, que se reciben y envían, desde y hacia el ordenador principal.

En este trabajo se describe el desarrollo conjunto de un hardware embebido y un firmware de puerta de enlace, implementado en un dispositivo intermediario, junto a un protocolo de transferencia de datos, que permite finalmente establecer el enlace entre el sistema de tiempo real duro (en adelante STR) y una computadora personal con un sistema operativo de tiempo diferido (en adelante SOTD).

Fundamentos

Sistemas de Tiempo Real

Un STR no siempre se refiere a un sistema de rápida respuesta, si bien una respuesta rápida puede satisfacer las demandas temporales del mismo. Tiempo real se refiere a que las tareas a desarrollar por el sistema tienen un tiempo de inicio y un vencimiento predeterminados, y que incumplir con los mismos tiene alguna consecuencia más o menos catastrófica. Los STR pueden tener restricciones de tiempo estrictas y no estrictas. Cuando el sistema posee el primer tipo, todo incumplimiento de la misma es considerado una falla catastrófica (por ejemplo: la aplicación tardía de refrigerante en un reactor nuclear, o la actuación tardía del sistema de frenado de emergencia automático en vehículos), y se lo denomina “de tiempo real duro”. Por otra parte si el incumplimiento de los límites en los tiempos de inicio o de vencimientos de las tareas no influye significativamente en el proceso se denomina al sistema “de tiempo real blando” (por ejemplo: reproducción de contenido multimedia, o voz sobre IP).

Sistemas Distribuidos

Los Sistemas Distribuidos pueden clasificarse por la forma en que se acoplan los elementos de procesamiento del mismo. Si lo que se tiene para intercambiar información entre los procesadores es una memoria compartida (como es el caso de los procesadores multinúcleo) el sistema distribuido es fuertemente acoplado. Si en cambio hablamos de un grupo de computadoras personales completas que comparten la información mediante una red (como es el caso de los clústeres de computadoras conectadas mediante

Ethernet) se dice que el sistema es débilmente acoplado. En todos los casos lo que se pretende es que el conjunto de computadoras o procesadores actúe como un servicio integrado destinado a un fin específico. Dichos sistemas están equipados con un software de sistemas distribuidos, el cual permite que las computadoras coordinen sus actividades y compartan recursos. El sistema que se aborda en el presente trabajo se define, más apropiadamente, como un conjunto de computadoras autónomas conectadas por una red, y con el software distribuido adecuado para que el mismo sea visto por los usuarios como una única entidad capaz de proporcionar facilidades de procesamiento.

El caso de estudio

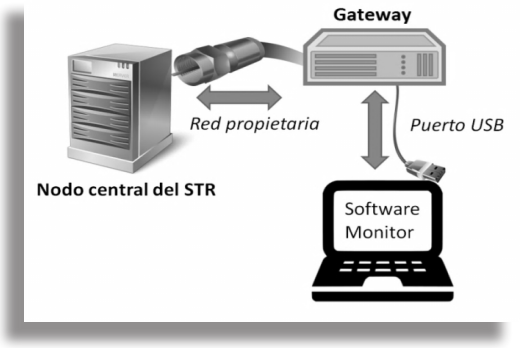
Teniendo en cuenta los requerimientos impuestos, se procedió a realizar un análisis de la factibilidad de reemplazar uno de los nodos preexistentes de la red por un emulador del mismo que corriera sobre una notebook. Se observó que, por su antigüedad y su empleo actual, sería imposible modificar el software o hardware de las computadoras que componen el sistema para integrar de manera nativa un nuevo enlace de comunicaciones externo. Sin embargo, se dispone del puerto de comunicación nativo con un protocolo propietario, por donde el ordenador central intercambiaba información con el nodo. En un trabajo anterior, miembros del equipo de investigación realizaron con éxito la re-ingeniería de la placa de comunicaciones original del sistema mediante dispositivos lógicos programables y a posteriori desarrollaron un sistema embebido que emula el handshake del mismo. De esta forma, se decidió desconectar el dispositivo que interactuaba con el ordenador central, para reemplazarlo por un dispositivo compuesto por un sistema embebido programable, microcontrolado con un Cortex M4.[5] Este dispositivo, llamado banco de pruebas, posee una interfaz compatible con las placas de comunicación, pudiendo enviar y recibir datos. La comunicación al ordenador central posee una estructura y semántica que debe res petar el periférico que se encuentre conectado al mismo, a fin de establecer una comunicación confiable a lo largo del tiempo. Esta comunicación es por

5 Características del microcontrolador Cortex M4. Sitio web de ARMDeveloper: https://developer.arm.com/products/processors/cortex-m/cortex-m4.

medio de tramas de entrada y salida de longitud fija. Respecto al temporizado, el sistema de tiempo real exige una transacción entrada/salida cada 500 ms.

Por otro lado, el banco de pruebas posee una interfaz USB, que permite la conexión con el dispositivo externo a un ordenador. En este caso, se seleccionó como dispositivo final una PC comercial, que ejecuta un software de generación y monitoreo de paquetes de la red cerrada.

Con el objetivo de que el intercambio de información sobre ambos extremos sea transparente, el protocolo junto con el algoritmo de enrutamiento de los datos, deberá diseñarse en torno al firmware del sistema embebido, confiriéndole al mismo la funcionalidad del *gateway,*[6] debido a su capacidad de operar interconectando dos canales de comunicación distintos. De esta forma, por un lado, la red de ordenadores interactuará naturalmente con nuestro dispositivo, debido a que se respeta la estructura de la comunicación que se tenía con el periférico anterior. En el extremo opuesto, se deberá poder establecer entre la PC y nuestro dispositivo, una comunicación asincrónica, libre de vencimientos, debiéndose definir inevitablemente al menos una trama de datos de entrada y una de salida. Un esquema de la conexión propuesta se muestra en la Figura 1.



Conexión preliminar

6 Definición de *gateway:* Telecommunications: Glossary of Telecommunications Terms. Editado por la National Telecommunication Information (1997).

Desarrollo

Diseño del *gateway*

Debido a que la cadencia de comunicación con el ordenador central está impuesta, el diseño del firmware del *gateway* parte de considerar una máquina de estados (en adelante FSM, del inglés Finite Estate Machine) que deberá realizar un ciclo cada 500 ms, en donde obligatoriamente se realice una transmisión y recepción de datos por la red propietaria. En caso de no existir datos de interés para el intercambio, se debe reenviar la última trama enviada o pasado un tiempo determinado, una trama vacía. Esto está previsto en la arquitectura de red propietaria, cumpliendo estas tramas un funcionamiento de señalización tipo *"keepalive"*.[7]

La comunicación entre el *gateway* y la PC se diseñó considerando que su interacción no podría cumplir con las exigencias del sistema de tiempo real. De manera que las etapas de transmisión y recepción serie, con la PC, se implementarán de forma distinta a las vinculadas al STR.

Dado que no fue posible definir un vencimiento comparable, se realizó un análisis de tiempos con el objetivo de acomodar las transacciones de la PC durante el tiempo ocioso del *gateway,* tiempo en el cual no se comunica con el otro ordenador. Estas operaciones deberían poder realizarse incluso de manera fragmentada, ocupando más de un ciclo de la FSM que se ejecuta en el *gateway.*

Definiremos como operaciones en *tiempo real* a aquellas que deben realizarse obligatoriamente en el plazo estipulado por el sistema de tiempo real, independientemente del estado de la PC. Las operaciones en *tiempo diferido* serán aquellas a las que no es posible aplicar un vencimiento, debido a que están vinculadas de una u otra manera a la PC.

Así, es posible definir una ventana temporal T_d para operaciones en tiempo diferido como la diferencia entre el tiempo de un ciclo de la FSM, y la duración de las comunicaciones que el sistema de tiempo real (STR) realiza durante ese lapso.

---

7 Keepalive Overview. Funciones de los paquetes de tipo "keepalive". The Linux Documentation Project. Disponible en: http://tldp.org/HOWTO/TCP-Keepalive-HOWTO/overview.

Durante ese tiempo, el *gateway* podrá hacer alguna de las siguientes tareas:

1. Procesar los datos recibidos desde la PC.
2. Procesar los datos recibidos desde el STR.
3. Realizar las comunicaciones pertinentes con la PC.

Queda claro que si se espera demasiado a que la PC transmita un conjunto de datos esperados, no será posible responder a las comunicaciones obligatorias del STR, generando un vencimiento. De esta forma, definiremos un tiempo máximo de espera dentro del ciclo de 500 ms. Vencido este tiempo, se deberá hacer ciclar obligatoriamente a la FSM, cumpliendo entonces con las comunicaciones de carácter obligatorio. Adicionalmente reservaremos tiempo para realizar las operaciones de proceso de datos, mencionadas anteriormente en los puntos 1 y 2.

Es por eso que dentro del T_d se deberán definir otras ventanas temporales, que llamaremos "ventanas de comunicación diferida" dentro de las cuales es seguro realizar las siguientes operaciones:

-Esperar a que la PC haya finalizado la transferencia de datos, para luego procesarlos inmediatamente.

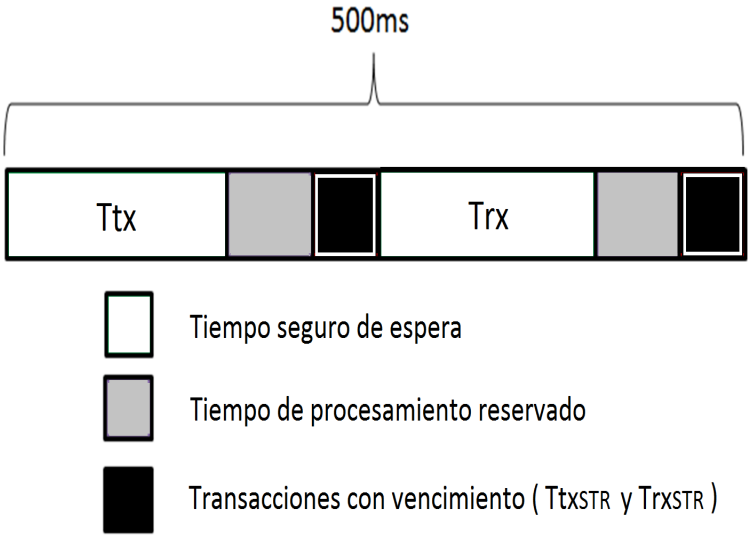
-Esperar a que la PC esté lista para recibir datos, para luego enviarlos inmediatamente.

Si repartimos equitativamente la duración de la ventana temporal tendremos, que el tiempo en el *gateway,* para recepción y transmisión diferida, queda expresado de acuerdo a las siguientes ecuaciones:

$$T_{tx} = \frac{T_d}{2} \text{ -Tiempo de proceso de datos a transmitir a PC} \quad (2)$$

$$T_{rx} = \frac{T_d}{2} \text{ --Tiempo de proceso de datos recibidos desde PC} \quad (3)$$

En la *Figura 2*, se muestra el esquema resultante de la distribución de las ventanas temporales dentro del ciclo de la FSM del *gateway*. En negro, se detallan las transacciones obligatorias, es decir, las que conllevan un vencimiento y deben ser cumplidas siempre. En blanco se detallan las ventanas temporales de comunicación diferida, y vinculado a estas, en gris se esquematiza el tiempo reservado para procesamiento en caso de existir transacción.



Distribución de las ventanas temporales dentro del ciclo de 500 ms de la FSM.

El desafío para lograr un enlace efectivo reside entonces en respetar siempre las transacciones con vencimiento pudiendo tolerar, en el otro extremo, comunicaciones que se desarrollen durante varios ciclos de la FSM. Esta limitación al diseño debe considerarse como una ventaja; dado que se aprovecha la robustez del sistema de tiempo real duro para diseñar en torno al mismo el comportamiento de la puerta de enlace y, consecuentemente, del protocolo de comunicaciones entre esta y la PC.

Para modelar la comunicación entre las partes, partimos del análisis de un flujo de datos que va desde la PC al STR, pasando por la puerta de enlace. Suponiendo entonces un caso en que nuestro sistema espera datos provenientes de la PC, se puede definir un conjunto de reglas de enrutamiento para lograr la funcionalidad buscada:

Regla 1. Si la PC no responde en el tiempo estipulado por la ventana temporal, se deberá habilitar una bandera que indique que hay una transacción diferida en curso, mientras se continúa con el funcionamiento de la FSM.

Regla 2. Se deberá esperar una determinada cantidad de ciclos de FSM a que la PC responda.

Regla 3. Durante los ciclos de máquina en que se espera a la PC, se reen-

viarán al STR los últimos datos recibidos.

Regla 4. Si se asume que la PC no responde, se deberá renegociar la conexión.

Regla 5. Se deberá estipular un vencimiento relativo a la retención de datos antiguos mencionada en la Regla 3. Una vez pasado este tiempo se realizarán las transacciones obligatorias con campos vacíos.

Regla 6. Mientras se espera la respuesta de la PC, no se podrán enviar datos provenientes del STR a la misma. Esto permite que ante una eventual sobrecarga, esta no se vea afectada con el cumplimiento de mayores demandas, permitiendo la descongestión de su buffer serie de recepción.

Diseño e implementación del protocolo de comunicaciones: transmisión de datos a la PC

El protocolo de comunicaciones con el que se estableció la comunicación PC - *gateway* se debió considerar en el software monitor ejecutado. Como se explicaba anteriormente, la implementación en la PC es transparente debido a que nuestro sistema debe operar correctamente en ambos dominios temporales.

En primer lugar, el envío de datos desde el *gateway* a la PC se realizó simplemente a través de una trama que contiene un encabezado y un terminador. Luego de recibir ambos, el software de la PC procesa la información contenida entre estas dos etiquetas, como muestra la Figura 3. Observando la Regla 6 detallada anteriormente, determinamos que esta trama se debe enviar únicamente cuando se compruebe que la PC está respondiendo a nuestros requerimientos.

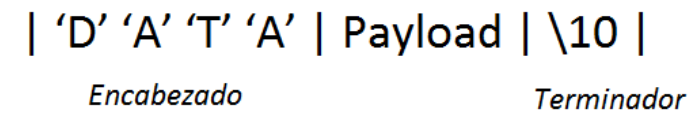


Figura 3 - Estructura particular de la trama implementada para enviar datos a la PC.

Por otro lado, no se implementó ningún mecanismo de espera que aguarde a que la PC esté lista para recibir. Se envía una sola vez la trama de datos completa, esperando que el sistema operativo de la PC y el software procesen a su tiempo la información que llega.

Diseño e implementación del protocolo de comunicaciones: recepción de datos de la PC

En principio, se podría diagramar la recepción de forma recíproca a la transacción descripta anteriormente, es decir, el *gateway* recibiría simplemente una trama, nuevamente delimitada por un encabezado y un terminador, y la procesaría cuando disponga de tiempo en la ventana temporal.

De esta forma, la PC es la que determinaría el momento en que ocurre la actualización de los datos que son enviados al STR. Este mecanismo, tipo "Peer to Peer" o de igual a igual, entre nuestro dispositivo y PC, presenta las siguientes desventajas:

- Una parte del ciclo de funcionamiento del *gateway*, y por ende, de la comunicación con el STR queda determinada por un sistema operativo que no posee restricciones temporales.
- La secuencia de envío y recepción que se debe mantener debe estar programada tanto en el *gateway* como en el programa de la PC, por lo que se requerirá de una instancia más de sincronismo de manera de determinar que efectivamente ambos extremos están listos para establecer una comunicación.
- El código se hace menos escalable dado que el control de la comunicación no queda centralizado.
- En caso de un error de transferencia no se podrá solicitar un reenvío de la información, debido a la imposibilidad de repetir una transacción, por ejemplo, un envío de datos sin antes haber completado la operación recíproca.

Por lo estipulado, se implementó, junto a la trama de recepción correctamente delimitada, un mecanismo de pedido de datos de parte del *gateway* a la PC, obteniendo el primero el control total sobre la comunicación, pudiendo cumplir con las seis reglas mostradas al principio mediante la implementación directa de las mismas en el código enrutador.

De esta forma, se solicita el envío de datos a la PC mediante un mensaje particular de control, que no forma parte de las tramas de entrada o salida. El extremo que ejecuta tiempo diferido no podrá enviar datos si no tiene una solicitud en espera de ser atendida.

El flujo de datos entre el *gateway* y la PC, que se desencadena mediante el protocolo definido, es finalmente esquematizado en la Figura 4, y descripto

a continuación.

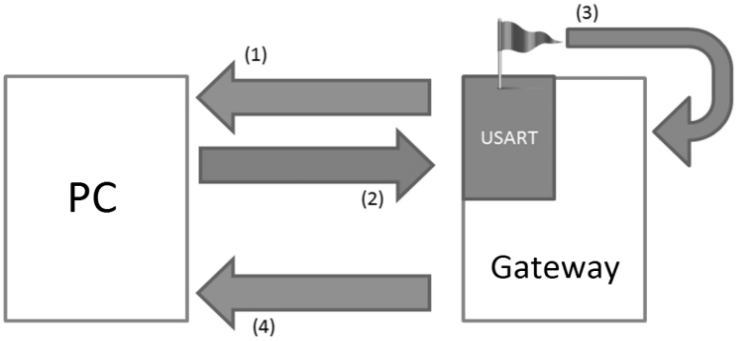


Figura 4 - Secuencia completa de comunicación PC - *gateway.*

1. El *gateway* envía la petición de datos a la PC.
2. La PC responde con una trama funcionalmente similar a la mostrada en el apartado anterior, que se almacena en un buffer de entrada relacionado a la interfaz serie de comunicación del *gateway.*
3. Únicamente cuando el buffer esté lleno, el UART del *gateway* levantará un flag (bandera), activando el procesamiento de datos. Este es el flag que marca el cierre de la ventana temporal vinculada a la recepción, T_{tx}.
4. Cuando los datos hayan llegado correctamente, entonces se sobreentiende que la PC está operativa, y se envía la información del STR a la PC. Como se estipuló anteriormente, esta operación no se realiza si alguna de las anteriores falla.

En la Figura 5, se muestra un esquema que integra la renegociación de la conexión con la PC, vinculada a la retención de datos mencionada en el apartado anterior. Para mejorar la claridad del mismo, se considerará el flujo de información que es generado en la PC y es enviado al STR.

En el punto 1, puede observarse que el STR es actualizado con los datos que envía el software monitor. En caso de que el software falle o tarde en responder, el *gateway* responderá a las transacciones obligatorias enviando los últimos datos recibidos por la PC (punto 2).

En el punto 3, se observa que luego de cierto tiempo de no respuesta, los datos antiguos son reemplazados por tramas vacías de *keepalive.* En el punto 4, el *gateway* intenta renegociar la conexión con la PC. Finalmente, en el

punto 5 esto se logra, actualizando el STR con nueva información generada por el software. Cabe aclarar que el mensaje de petición de datos se lo señala como “SEND/RQ”.

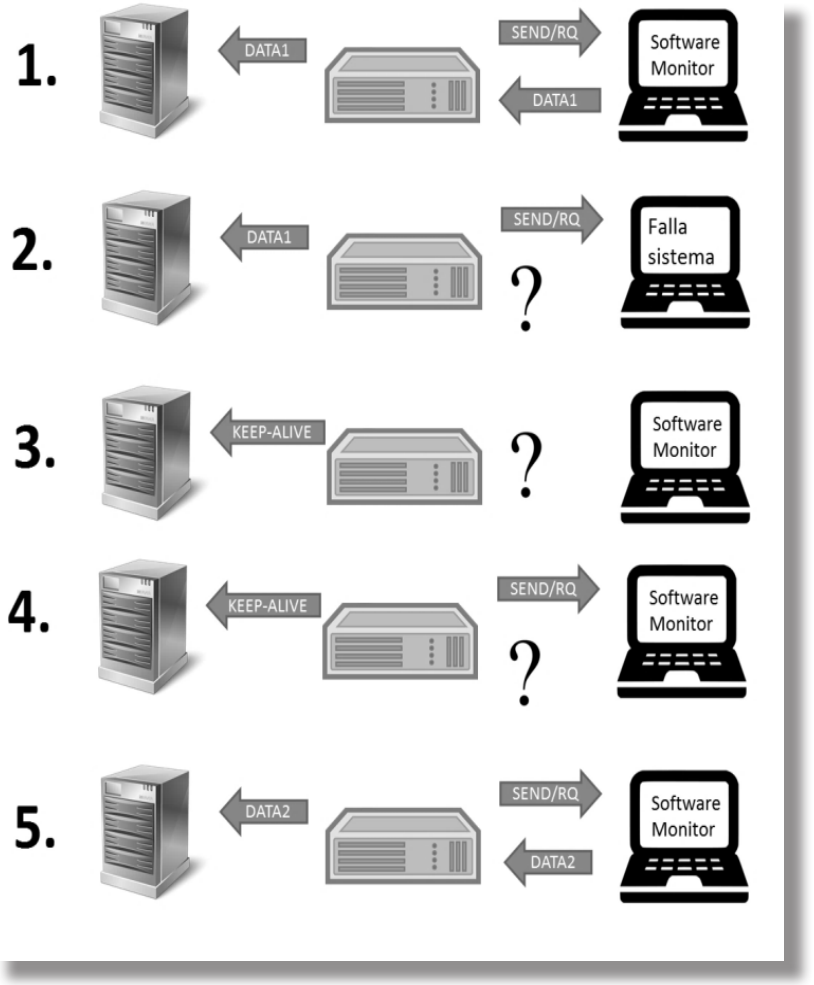


Diagrama temporal de renegociación de la conexión.

Manejo de los errores de transferencia

Cuando se consideran las seis reglas implementadas anteriormente, se puede determinar que durante el funcionamiento normal del programa, el buffer de entrada en la PC puede contener las tramas esquematizadas en la

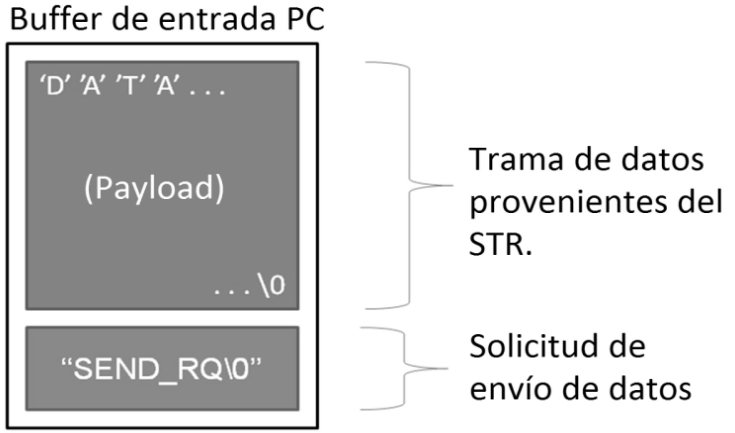


Figura 6 - Representación del contenido de buffer que normalmente presenta la PC.

Esto es debido a que no se mandarán tramas de datos hasta que no se hayan procesado los requerimientos de envío por parte de la PC, de manera que, en condiciones normales, siempre existirá una trama de datos del STR alternada con un comando de solicitud proveniente del *gateway.*

Una de las funciones del programa de la PC es justamente reconocer estas tramas para procesarlas y eliminarlas del buffer, con el objetivo de reconocer la próxima instrucción o trama, liberando espacio para posteriores recepciones.

Un problema puede ocurrir si la PC no responde en el tiempo adecuado a la solicitud, quedando esta almacenada en su buffer de entrada. Si pasa el tiempo suficiente, el *gateway* asumirá, de acuerdo a la Regla 4, que esta instrucción nunca llegó, enviándola nuevamente. Se muestra en la Figura 7 el estado de ambos nodos, durante este caso de error.

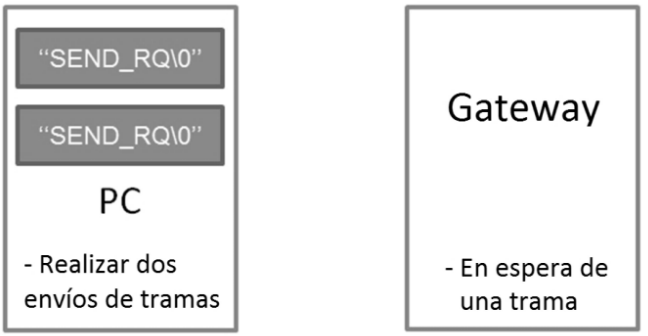


Figura 7 - Diacronía de comandos.

En este momento se presenta un caso de diacronía: el *gateway* recibirá los datos esperados, pero el programa no tiene en cuenta que existe una transacción pendiente que puede darse en cualquier momento.

Siguiendo el temporizado propuesto, puede probarse que la PC enviará tramas corruptas al *gateway*. De esta manera, implementando en el sistema una estrategia de descarte de paquetes erróneos, puede recuperarse el sincronismo con la PC. Bastará enviar una nueva petición para que se establezca una comunicación coherente entre ambos extremos.

Cabe aclarar que cuanto mayor sea el tiempo de espera de respuesta, menor será la posibilidad de que este error ocurra. Sin embargo será más lenta también la reconexión con la PC.

Otro caso de error puede ser el que se presenta directamente cuando se recibe una trama que no es de la longitud correcta. Ergo, el buffer no se llena por lo que no se levanta su flag señalizador de llenado (visto en la Figura 5) y se considera que la PC no responde a tiempo. Cuando se cumple el vencimiento de espera de respuesta de la PC, debe resetearse el buffer de entrada, a fin de dar una nueva posibilidad a la PC para que envíe la cantidad de datos correctos.

El problema inverso surge de pensar que la PC envía más datos de los que se esperan. En ese caso, cuando el buffer se llena, no se reciben más datos quedando estos excluidos de la trama que se leerá. Con la implementación del descarte de paquetes erróneos se podrá verificar fácilmente la inexistencia del terminador en la trama.

Ensayo del *gateway*

La prueba de funcionamiento del protocolo y *gateway* se realizó practicando el esquema especificado en la Figura 1. La PC puede permanecer encendida en todo momento independientemente del estado de los otros actores en la comunicación.

En primer lugar, se encendió el STR y se esperó que entre en sincronismo con todos los periféricos, controlando las variables que interesa monitorear externamente. A continuación se encendió el *gateway* y se verificó que se intercambiaran correctamente tramas de *keep alive* desde y hacia el sistema de tiempo real.

La conexión de la PC mediante el puerto serie, que es controlado por el software de monitoreo, puso en marcha el protocolo de comunicación entre este y el *gateway.* Se comenzaron a monitorear en el software las primeras variables de control enviadas desde el STR. El mismo presenta la posibilidad de cargar datos en la trama de salida desde la PC al *gateway*, por lo que se generaron mensajes que fueron interpretados correctamente en el otro extremo.

Con el objetivo de probar la correcta aplicación de las reglas definidas anteriormente, se realizaron pruebas de conexión y desconexión de la PC al *gateway* durante el funcionamiento del mismo, simulando fallas en el sistema operativo de la PC. Para esta prueba se comprobó que luego de establecer el enlace PC - *gateway,* la desconexión de la primera no impedía que durante un tiempo el STR siguiera recibiendo los últimos datos enviados por la PC, para luego comenzar a recibir tramas vacías, como se mostraba en Figura 6.

Por otro lado, durante las pruebas, se generaron errores en el programa ejecutado sobre la PC, el cual sostenía una comunicación correcta con el *gateway* por un período de un minuto aproximadamente para luego generar una excepción en el handler del puerto serie, vinculado a un time out, perdiendo la comunicación, e impidiendo que el *gateway* inicie la renegociación de la conexión. Esto, posiblemente, es causado por el driver del USB de la PC que podría llegar a causar inconvenientes cuando se sostienen comunicaciones frecuentes de tipo ráfaga.

Aprovechando la robustez del protocolo de comunicación con *gateway,* se implementó una rutina de catch,[8] en el software de alto nivel de la PC, que

desencadena un descarte del buffer del puerto serie, generando del lado del *gateway* un caso de diacronía como el comentado anteriormente, que es imediatamente solucionado, reestableciendo la conexión sin impactar negativamente en el desempeño de la comunicación.[9]

Conclusiones

La implementación del *gateway* con un sistema embebido microcontrolado resultó una solución eficaz, de bajo costo y segura debido al diseño de la FSM, que fue programada en bare metal.[10]

Si bien la solución en particular no es escalable a sistemas en donde la cadencia de comunicación sea demasiado rápida, el diseño de la FSM del *gateway,* junto con el protocolo de comunicación diferida, supone un conjunto de prácticas convencionales que pueden ser portadas a otras arquitecturas de software o hardware para lograr, al menos, una funcionalidad mínima. Es conveniente aseverar que este trabajo no proporciona una solución genérica para el traslado de información entre sistemas con distintos dominios temporales. Siempre se deben considerar las características puntuales del STR con el cual se desea establecer una comunicación bidireccional.

Finalmente, la consideración de las exigencias del STR durante la etapa del desarrollo dio como resultado un *gateway* robusto y de fácil depuración,

debido a que las comunicaciones diferidas fueron diseñadas en torno a las que poseen vencimientos. Las limitaciones de diseño, una vez que fueron correctamente especificadas, permitieron enfocar la generación de soluciones en los puntos críticos del desarrollo.

---

8 Rutinas de Try/Catch. Documentación de C# para Microsoft Windows. Sitio web de Mi crosoft Docs. Disponible en: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/try-catch.

9 Sincronizando un buffer mediante la técnica de descarte. Microsoft MSDN. Sitio web: https://msdn.microsoft.com/en-us/library/system.io.streamreader.discardbuffereddata(v=vs.110).aspx.

10 Ejemplos de aplicaciones microcontroladas ejecutadas en bare-metal. Extraído de la Wiki de procesadores de Texas Instruments. Sitio web: http://processors.wiki.ti.com/index.php/Processor_SDK_Bare_Metal_Examples#ARM_Coretex-M4.

Bibliografía

Caspe, Franco S., Pita, Emmanuel, Galasso, Christian L., Banchieri, Miguel A. (2016). Plataforma de pruebas para interfaces de red en tiempo real basado en un sistema embebido. Congreso Argentino de Sistemas Embebidos. ISBN 978-987-45523-8-9.

Coulouris, G., J. Dollymore y T. Kindberg (2001). *Distributed Systems Concepts and Design*. Addison-Wesley, 3rd edition.

Galasso, Ch. L.; Friedrich, G. R.; Burgos, S. O.; Díaz, G. J.; Antonini, A. A. (2013). Reingeniería de las interfaces entre computadoras del sistema de comando y control. *Revista Digital del INUN,* N° 4. Año 2013. Dra. Lucía Alejandra Destro, Dra. Diana Fernández Calvo (eds.), pp. 59-98. ISSN 1853-4015. ISSN: 1852-7205. URL: http://www.ara.mil.ar/archivos/Docs/Galasso_corregido1.pdf

Intermediary network devices. Definición disponible en: http://www.ciscopress.com/articles/article.asp?p=2158215&seqNum=6.

Koshy, Thomas. Discrete Mathematics with Applications, pp. 733-802. ISBN: 0-12-421180-1.

Oliver, Juan Pablo (2007). Tesis de Maestría en Ingeniería Eléctrica, pp. 5-8. ISSN: 1510-7264 https://iie.fing.edu.uy/publicaciones/2007/Oli07/Oli07.pdf.