Sistemas Distribuidos de Tiempo Real

Apuntes: TEMA 2

Por: J. Javier Gutiérrez gutierjj@unican.es
http://www.ctr.unican.es/

Grupo de Computadores y Tiempo Real, Universidad de Cantabria

Sistemas distribuidos de tiempo real

PARTE I: Distribución y tiempo real

- **TEMA 1. Conceptos básicos de distribución y tiempo real**
- **TEMA 2. Comunicaciones de tiempo real**

Comunicaciones de tiempo real

Normalmente el problema de la expulsión se resuelve partiendo los mensajes en paquetes de una longitud máxima fija:

- **se convierten en la unidad mínima no expulsable**
- **su tiempo de transmisión se contabiliza como un bloqueo**

Vamos a ver dos protocolos de comunicaciones basados en prioridades que garantizan tiempo real:

- **Bus CAN**
- **RT-EP**

Introducción al Bus CAN

El Bus CAN (Controller Area Network) es un protocolo de comunicaciones desarrollado por la firma alemana Robert Bosch GmbH para automoción:

- topología de bus
- transmisión serie de mensajes de hasta 8 bytes
- velocidad de transmisión de hasta 1 Mbit/s
- arbitrio del bus basado en prioridades (tiempo real)
- pertenece al grupo de buses de campo utilizados en control distribuido
  - control de sensores y actuadores distribuidos
  - uso de dispositivos inteligentes que disminuye el cableado
- se usa con protocolos de alto nivel

Características del bus CAN

Mensajes

- la información se envía en mensajes con un formato fijo
- puede tener tamaños de datos entre 0 y 8 bytes

Encaminamiento de la información (*routing*)

- no existe identificación de nodos, ni direcciones, lo que implica:
  - los nudos pueden entrar y salir del sistema sin que se requiera una nueva configuración
  - los mensajes se identifican por el valor de su prioridad, y aunque ésta no indica el destino, se pueden aplicar filtros en cada nodo para decidir qué mensajes se aceptan
  - la información se puede enviar simultáneamente a varios nodos (*multicast*)
  - la información la reciben todos los nodos o ninguno (consistencia)

Características del bus CAN (cont.)

Velocidad de transmisión

- se puede elegir hasta el máximo de 1 Mbit/s
- debe ser uniforme y fija para todos los nodos conectados

Arbitrio

- se realiza por el identificador del mensaje que constituye su prioridad
- se utiliza durante el acceso al bus
- cualquier nudo puede empezar a transmitir en cualquier momento si el bus está libre (*multimaster*)

Características del bus CAN (cont.)

Seguridad

- detección de errores
  - monitorización: el transmisor compara el valor del bit transmitido con el que encuentra en el bus
  - chequeo de redundancia cíclica (CRC)
  - *bit stuffing* (se añaden unos o ceros extra)
  - chequeo de marco de mensaje
- la probabilidad de no detectar un error es del orden de 10^{-11}

El canal físico normalmente consiste en un par de cables diferenciales, pero puede ser también una fibra óptica

Características del bus CAN (cont.)

Valores del bus

- el bus puede tomar dos valores lógicos complementarios:
  - dominante o recesivo
- en caso de transmisión simultánea impera el dominante
- por ejemplo, si se implementa como una AND cableada el dominante es el 0 y el recesivo el 1
- no se dan valores físicos en la especificación

Arbitrio del bus CAN

Cuando el bus está libre cualquier nodo puede comenzar a transmitir un mensaje

Si dos o más nudos comienzan la transmisión a la vez, el conflicto se resuelve bit a bit por el identificador del mensaje (su prioridad):

- los bits del identificador se transmiten uno a uno desde el más significativo
- cada transmisor compara el bit que ha puesto con el que hay en el bus
  - si es el mismo continúa la transmisión del siguiente
  - si ha puesto un recesivo y lee un dominante se retira hasta que el bus quede libre de nuevo

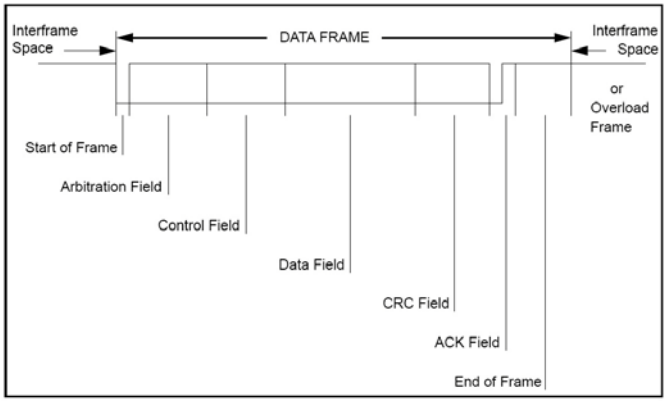
Arbitrio del bus CAN (cont.)

No debe haber dos mensajes con el mismo identificador

Hay dos especificaiones de CAN:

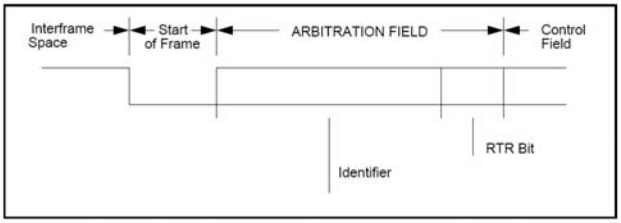
- **CAN 2.0 A: identificador de 11 bits**
- **CAN 2.0 B: identificador de 29 bits**

Formato de mensaje CAN



Bosch

Campo de arbitrio



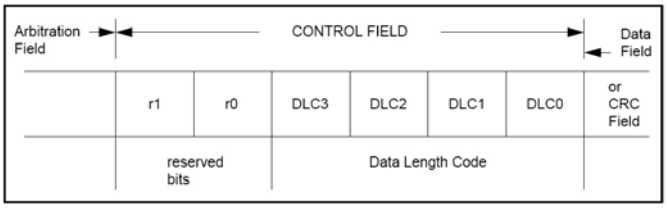
Bosch

RTR: bit de requerimiento de transmisión remota

Prioridad: los bits se transmiten desde el más significativo

- **para el identificador de 11 bits los 7 más significativos no pueden ser todos recesivos**

Campo de control

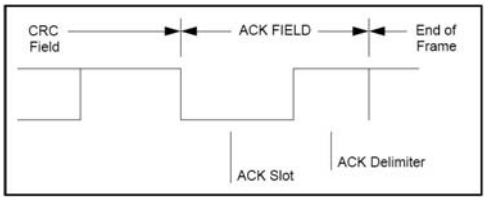


Bosch

El campo de control tiene 6 bits

- **los bits DLC portan la longitud del mensaje en bytes con valores de 0 a 8**

Campos de CRC, ACK y fin



Bosch

El CRC es una secuencia de 15 bits con el código calculado

El final del marco del mensaje consta de 7 bits recesivos consecutivos

Campos de CRC, ACK y fin

El campo de reconocimiento consta de dos bits que el transmisor los pone como recesivos:

- **ACK SLOT: si el receptor recibe el mensaje correctamente lo ponen como dominante**
- **ACK Delimiter: se deja recesivo**

Comparación con otros buses

	CAN	LIN	I2C
Compañía que lo desarrolló	Bosh	Open source	Philips
Velocidad	1Mb/s	20 Kb/s	0.1Mb/s / 0.4Mb/s
Tamaño de datos	64bits	8bits	8bits
Prioridad de mensajes	Si	No	No
Garantía de latencia	Si	***	No
Flexibilidad en la configuración	Si	***	Si
Sistema Multimaestro	Si	No	Si
Detección y señalización de errores	Si	Si	Si
Retransmisión de tramas	automática	No	programable

Scott Mueller,"Upgrading and Repairing PCs", 17th Ed., QUE, 2006

Protocolos de alto nivel sobre CAN

Se basan en el uso del identificador con un fin especial:

- **CANKingdom**
  - define primitivas de protocolo basadas en CAN
  - se define un master del bus en la inicialización que chequea los nodos que hay conectados a la red
  - soporta modelos gobernados por eventos y por tiempo
- **OSEK-COM**
  - principalmente usado en automoción
- **SDS (*Smart Distributed Systems*)**
  - define el nivel físico y el de aplicación
  - usado por sensores y actuadores inteligentes sobre un cable de 4 hilos
  - admite hasta 64 nodos y 126 direcciones

Protocolos de alto nivel sobre CAN (cont.)

- **DeviceNet**
  - implementa otro protocolo (***Common Industrial Protocol***) sobre CAN
- **CANOpen**
  - implementación de objetos basados en los identificadores de mensaje
  - permite controlar dispositivos e intercambiar datos a través de los diferentes tipos de objetos que define
- **TTCAN (*Time Triggered CAN*)**
  - permite el uso determinista del bus CAN

Cada protocolo de alto nivel requiere dispositivos especiales para que los puedan controlar

El bus CAN y el tiempo real

El bus CAN es una red capaz de transmitir mensajes en tiempos acotados

- al basarse en prioridades fijas se puede aplicar el análisis de planificabilidad similar a los procesadores para obtener cotas de los tiempos de transmisión de los mensajes que genera una aplicación
- se pueden aplicar técnicas de análisis de sistemas distribuidos
- para que esto sea posible los protocolos de alto nivel que operan sobre el bus tienen que haber sido diseñados para hacer un uso adecuado de las prioridades

El bus CAN y el tiempo real (cont.)

Por ejemplo, un protocolo de nivel de aplicación podría utilizar el identificador de mensaje como sigue:

	Identifier										
Bit Number	11	10	9	8	7	6	5	4	3	2	1
Field	Message Priority				Destination Node			Communication Channel			

- Prioridad:
  - urgencia del mensaje
  - independiente de los nodos y la funcionalidad
  - mapea las prioridades
- Nodo destino:
  - al menos debe haber un nodo por cada controlador de CAN en el sistema
  - utiliza el filtrado de mensajes

El bus CAN y el tiempo real (cont.)

- Canal de comunicaciones:
  - identifica la tarea o thread destinatario
  - la tarea elige el canal del que quiere leer, o en el que espera la recepción de mensajes
  - papel similar al puerto de comunicaciones

Con el identificador de 29 bits las posibilidades de codificación aumentan considerablemente

Para comunicaciones se debe usar un mecanismo de particionamiento de mensajes mayores de 8 bytes en paquetes

Introducción a RT-EP

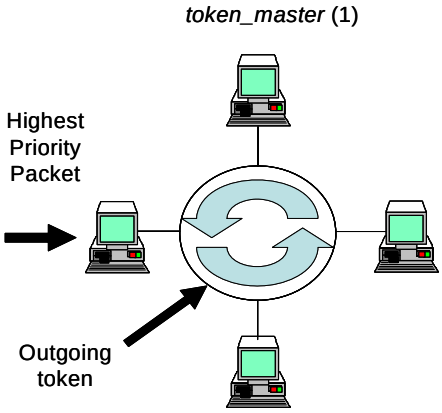
RT-EP (Real-Time Ethernet Protocol) es un protocolo de paso de testigo basado en prioridades:

- el testigo circula sobre un anillo lógico:
  - número de estaciones fijo
  - cada nudo conoce la dirección de su sucesor
  - un nudo arbitrario crea el testigo y se convierte en *token_master*
  - el *token_master* es dinámico
  - el mensaje se parte en paquetes
- la comunicación se realiza en dos etapas:
  - arbitrio de prioridad
  - transmisión del mensaje de datos

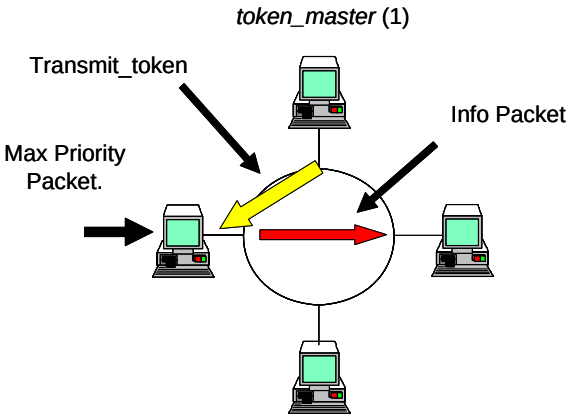
Introducción a RT-EP (cont.)

- arbitrio de prioridad:
  - el testigo viaja por el anillo visitando todos los nudos
  - cada estación chequea y actualiza si uno de sus paquetes tiene mayor prioridad
  - el testigo se envía al nudo sucesor
  - el proceso se realiza hasta que el testigo llega al *token_master*
- transmisión del mensaje de datos:
  - el *token_master* envía un mensaje a la estación con el paquete de mayor prioridad que tiene permiso para enviar
  - se envía el paquete con los datos
  - la estación receptora se convierte en *token_master*

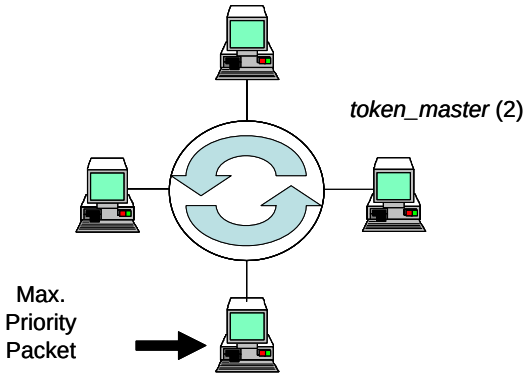
Arbitrio de prioridad



Transmisión del mensaje



Nuevo arbitrio de prioridad



Manejo de fallos

Se consideran los siguientes fallos:

- **Fallo de estación: se reconfigura el anillo**
- **Pérdida de paquete: se retransmite de nuevo**
- **Duplicidad de paquete: se elimina el paquete**

Se garantiza tiempo real en la pérdida de un paquete

Implementación:

- **Cada estación escucha el medio después de enviar un paquete hasta su reconocimiento o un timeout:**
  - **el reconocimiento es la transmisión de un marco**
- **Para eliminar paquetes duplicados se usa un número de secuencia**

Formatos de RT-EP

Marco Ethernet II

8 bytes	6 bytes	6 bytes	2 bytes	46-1500 bytes	4 bytes
Preamble	*Destination Address*	*Source Address*	*Type*	Data	*Frame Check Sequence*

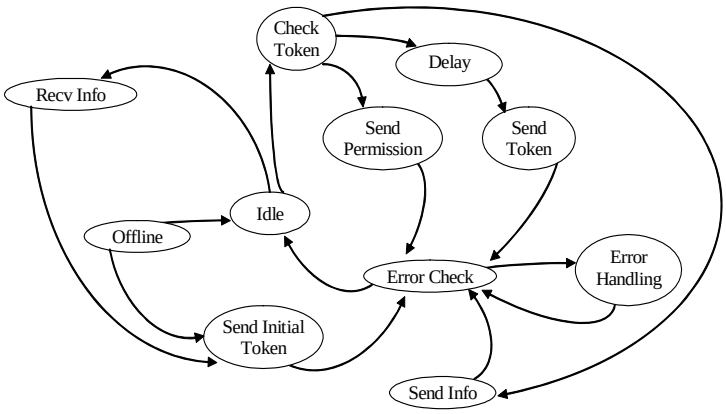
Paquete de testigo: usado para transmitir el token

1 byte	1 byte	2 bytes	6 bytes	2 bytes	6 bytes	6 bytes	22 bytes
Packet Identifier	*Priority*	*Packet Number*	*Token_Master Address*	*Failure Station*	*Failure Address*	*Station Address*	*Extra*

Paquete de información: usado para transmitir datos

1 byte	1 byte	2 bytes	2 bytes	2 bytes	0-1492 bytes
Packet Identifier	*Priority*	*Packet Number*	*Channel ID*	*Info Length*	*Info*

Máquina de estados de RT-EP



Funcionalidad de RT-EP

Cada estación tiene una cola de transmisión (cola de prioridad)

- los paquetes se encolan ordenados por su prioridad

El tamaño de la información en un paquete se limita a 1492 bytes y en este nivel no hay fragmentación

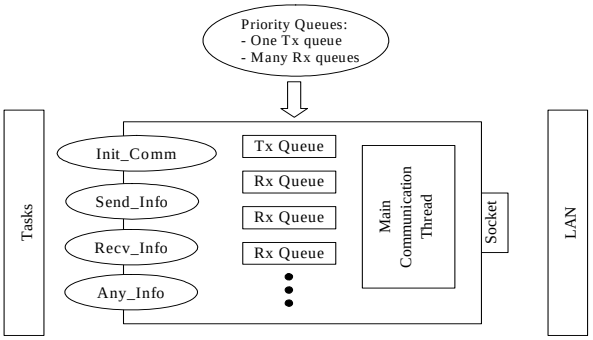
Cada estación tiene un conjunto de colas de prioridad de recepción

- el número es configurable

Cada thread de la aplicación puede tener asociada su propia cola de recepción

- la aplicación debe asignar un número (Channel_ID) a cada thread que se comunique a través del protocolo

Funcionalidad de RT-EP (cont.)



Interfaz del protocolo: envío

```
--  Sends Data over the network to a specific Channel in a the
--  Destination Address (could be multicast) at a given Priority
--  or through a Server
--  Exceptions: Station_Not_Valid, Station_Not_Found,
--  Invalid_Channel, Unexpected_Error, Info_Length_Overflow

--  Send at a given Priority
procedure Send_Info
     (Destination_Station_ID : in Station_ID;
      Channel_ID             : in Channel;
      Data                   : in Stream_Element_Array;
      Data_Priority          : in Priority);
```


Interfaz del protocolo: envío (cont.)

```
--  Send through a given Server
procedure Send_Info
     (Destination_Station_ID : in Station_ID;
      Channel_ID             : in Channel;
      Data                   : in Stream_Element_Array;
      Id                     : in Server_ID;
      Blocking               : in Boolean := False);
```

Interfaz del protocolo: recepción

```
--  Reads from the Channel_ID the message with highest priority
--  Exceptions: Invalid_Channel, Unexpected_Error

--  Blocking version with Streams
procedure Recv_Info
     (Source_Station_ID : out Station_ID;
      Channel_ID        : in Channel;
      Data              : out Stream_Element_Array;
      Last              : out Stream_Element_Offset;
      Data_Priority     : out Priority);
```


Interfaz del protocolo: recepción (cont.)

```
--  Non-Blocking version with Streams (if No Elements,
--  Received is false)
procedure Try_Recv_Info
     (Source_Station_ID : out Station_ID;
      Channel_ID        : in Channel;
      Data              : out Stream_Element_Array;
      Last              : out Stream_Element_Offset;
      Data_Priority     : out Priority;
      Received          : out Boolean);
```


Interfaz del protocolo: recepción (cont.)

```
--  Blocking version Generic
generic
      type Data_Type is private;
procedure Generic_Recv_Info
      (Source_Station_ID : out Station_ID;
       Channel_ID        : in Channel;
       Data              : out Data_Type;
       Data_Priority     : out Priority);
```

Interfaz del protocolo: recepción (cont.)

```
--  Non-Blocking version Generic (if No Elements,
--  Received is false)
generic
      type Data_Type is private;
procedure Generic_Try_Recv_Info
     (Source_Station_ID : out Station_ID;
      Channel_ID        : in Channel;
      Data              : out Data_Type;
      Data_Priority     : out Priority;
      Received          : out Boolean);

--  Any_Info checks if there is any data to be received
--  in a specific Channel
function Any_Info (Channel_ID : in Channel) return Boolean;
```


Configuración de RT-EP

Ficheros a modificar:

- **rtep-protocol-stations-ring_spec.ads**
  - configuración del anillo: mac y nombre de las estaciones
- **rtep.ads**
  - tiempo de inicialización
  - tiempo de espera (control de sobrecarga)
  - número de estaciones en el anillo
  - límite de tiempo para hacer retransmisión
  - número de retransmisiones

Modelo para el análisis del protocolo

En el modelo para el análisis es necesario evaluar:

- las sobrecargas en las CPUs (debidas al código del driver)
  - las diferentes operaciones implicadas se pueden observar en la máquina de estados
- el bloqueo que se puede sufrir en la red como consecuencia de que el paquete no es expulsable; constituido por:
  - dos rotaciones del testigo
  - más el envío de un paquete

Ejemplo de sobrecargas en la CPU para MaRTE OS

Medidas en dos PCs (Pentium III 700 MHz) con MaRTE OS y Ethernet a 10 Mbps

RT-EP CPU Overheads	Worst (μs)	Best (μs)	Av (μs)
Idle State (+ Error_Check)	48.35	10.8	11.16
Send_Initial_Token	41.16	30.85	31.44
Check_Token	18.51	9.32	9.45
Send_Permission	25.93	24.40	24.75
Send_Token	41.39	24.30	24.74
Send_Info	41.63	37.44	38.36
Recv_Info	37.21	21.19	21.9