Diseño y desarrollo de un sistema teleinformatico Experimental

Roberto Cardenas

RESUMEN

Se describe en este artículo el desarrollo de la plataforma física y software para una red LAN. Se escoge la interfaz RS-485, con la cual se pretende crear una plataforma que permita la experimentación de protocolos, así como la implementación práctica de sistemas tele-informáticos. Para ello, se desarrolla un transceiver que convierte el puerto serial RS-232 en un puerto controlador de red que tiene señales de transmisión, recepción, colisión, supervisión y control.

Palabras Claves: Máquina de estados finitos, RS-485, transceiver, puerto serial, maestro, esclavo, sondeo, selección, crc, objeto.

ABSTRACT

It is described in this article the physical platform and the software development for a net LAN. The interfaz RS-485 is chosen, with which is sought to create a platform that allows the experimentation of protocols, as well as the practical implementation of tele-computer systems. For it, a transceiver is developed that converts the serial port RS-232 in a port net controller that has transmission signs, reception, collision, supervision and control.

Key Words : Machine of finite states, RS-485, transceiver, serial port, master, slave, poll, selection, crc, object.

INTRODUCCION

El desarrollo de software en sistemas teleinformáticos requiere en algunos casos de implementaciones particulares dadas las restricciones de carácter comercial. Una plata Forma física ha sido desarrollada para una red LAN. La selección de la interfaz RS-485 [1], con la cual se crea una plataforma que permita la experimentación de protocolos, así como la implementación práctica de sistemas tele-informáticos. Los procesos del nivel de enlace de datos se realizan mediante rutinas escritas en lenguaje C residentes en la memoria de cada estación del sistema [2]. Además se desarrolla una máquina de estados finitos así como el protocolo que permite la comunicación. En el nivel de aplicación se implementa un controlador de terminales tanto maestras como esclavas. El problema es planteado de desarrollo de hardware, software e integración del sistema. Bajo esta óptica, adicional al transceiver como plataforma física, el hilo conductor que cohesiona todo el prototipo tele-informático es un objeto llamado SIO (serial input output) en donde se encuentra la descripción tanto de datos como de código del sistema.

El problema es planteado de desarrollo de hardware, software e integración del sistema.

EL ESTANDAR RS-485

La idea básica del proyecto desarrollado es convertir el puerto RS-232 en un puerto de red de datos multipunto. Este puerto no puede ser usado directamente para este propósito por las siguientes razones: velocidad limitada, longitud limitada, suceptibilidad al ruido, no está concebida como una norma multipunto. En una interfaz RS-485 la información se transfiere a una línea de transmisión balanceada [3].

ARQUITECTURA DEL TRANSCEIVER

El transceiver tiene lo siguientes bloques funcionales (ver Figura 1):

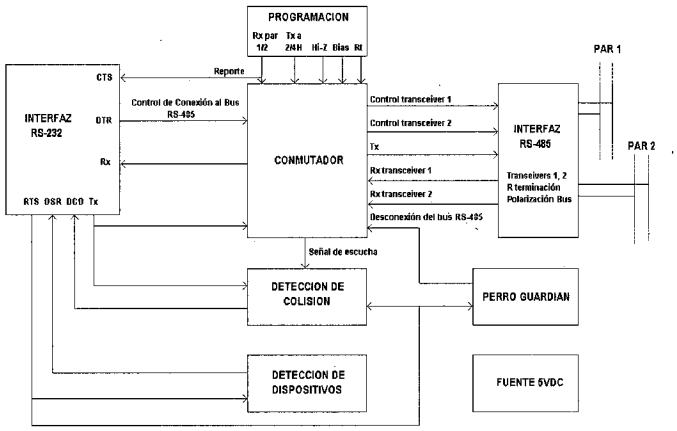

Figura 1. Diagrama en bloques del Transceiver.

Interfaz RS-232: deben convertir la señales eléctricas provenientes del puerto RS-232 del PC a señales TTL. Desde el transceiver hacia el PC se hace la operación inversa [4], algunas funciones son:

Conmutador: tiene la función de conectar los pares de transmisión y recepción de acuerdo a lo seleccionado en el bloque de programación.. Este bloque está implementado con el multiplexor 74HC157 [5].

Detector de Colisión: hace una comparación con una compuerta O Exclusiva, de la señal de datos que se está transmitiendo sobre el canal y la señal de datos producida por el PC en niveles TTL

Perro Guardián: Esta función es implementada con el contador 4541 [6].

Interfaz RS-485: en el diseño existen dos transceivers RS-485 (implementados con integrados DS75176), ya que existe la posibilidad de funcionar a dos o cuatro hilos.

Programación: funcionamiento a dos o cuatro hilos, polarización del bus, resistencias de final de línea.

Detección de Dispositivos: este módulo se encarga de dar la señal a la estación indicando que se está demandando los servicios del transceiver por parte de la estación. Igualmente, la estación detecta, con este módulo, que tiene un transceiver conectado.

Fuente: consta de un adaptador de 110/9 VAC y un regulador en la tarjeta de 5VDC de salida.

ORGANIZACION DEL SOFTWARE

Para funcionar en un ambiente de red, fué necesario desarrollar los niveles 2 y 7 del modelo OSI. La organización detallada del software y el diseño de hardware se puede ver en [7].Tanto para la estación Master como para las estaciones Slave la información está dividida en listados .h (encabezados), donde se definen las constantes y estructuras del sistema y de algún modo el vocabulario que se va a usar en la programación.

Por otro lado están los archivos .lib. Hablando rigurosamente estas no son unas librerías estándar, ya que no están en código objeto. Símplemente son una colección de rutinas agrupadas por algun concepto.

NIVEL DE APLICACIÓN

El manejo del dispositivo de entrada/salida de red (UART-TRANSCEIVER), se realiza por un método de sondeo, con algunas consideraciones para no perder información. La idea más importante tiene que ver con el almacenamiento de los estados de las variables de los terminales en zonas de trabajo. Esto sugiere el uso de variables llamadas objetos. Para el sistema, como se verá, se diseñó un objeto que no solo contendrá la información del nivel de aplicación, sino de todo el sistema.. Una parte del código se muestra en "ACTUALIZAR TERMINAL ACTIVO", cómo se hace la actualización de datos del terminal que se está tratando en el MASTER (serv):

ACTUALIZAR TERMINAL ACTIVO

```
.
.
if (serv->internal_term_addr == INT_TERM1_ADDR)
 siop->term1 = siop->server;
if (serv->internal_term_addr == INT_TERM2_ADDR)
 siop->term2 = siop->server;
if (serv->internal_term_addr == INT_TERM3_ADDR)
siop->term3 = siop->server;

 /************************************************/
```

El servidor tanto como los terminales tienen una estructura de datos idéntica llamada *terminal.* Los punteros hacia las estructuras que son *terminal* están en el objeto SIO (*serv, term1, term2, term3*).

El código de aplicación es re-entrante. Esto significa que solo el terminal (term1, term2 o term3) que se esté atendiendo en el MASTER cargará sus parámetros en serv, que es la estructura de servicio en el MASTER.

La entrada de datos desde el teclado se hace por la función entry(), la cual se ejecuta una vez en cada ciclo del programa principal.

En la pantalla de MASTER aparecen seis bloques donde se pueden desplegar datos. En los tres superiores se pueden escribir mensajes (máximo de 128 caracteres) que van hacia los terminales 1, 2 y 3 respectivamente. En los tres bloques inferiores se despliegan los datos procedentes de los terminales 1, 2 y 3 respectivamente (ver Figura 2).

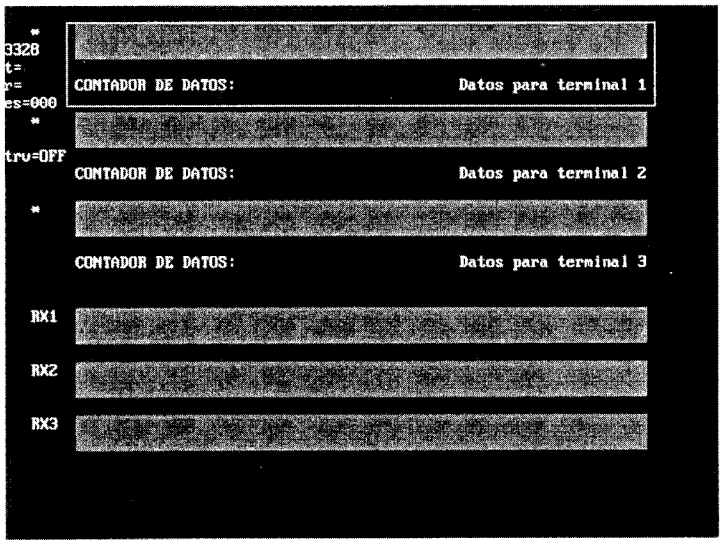

Figura 2. Despliegue en el MASTER.

NIVEL DE ENLACE DE DATOS

Esta capa tiene como función prestar un servicio bien definido al nivel de RED (en este caso no existe), determinar cómo los bits del nivel FISICO están agrupados en tramas, manejar los errores de transmisión y regular el flujo de tramas de manera que receptores lentos no sean "ahogados" por transmisores rápidos.

El protocolo de comunicaciones que se implementó solo hace uso de este nivel. El protocolo es del tipo sondeo. En esta modalidad un terminal MAESTRO o host interroga a estaciones ESCLAVAS. El campo DIRECCION siempre tendrá la dirección de la estación esclava.

El protocolo de sondeo es determinístico, lo que significa que no hay lucha por alcanzar el canal (contención)

El protocolo de sondeo es determinístico, lo que significa que no hay lucha por alcanzar el canal (contención), si no que el acceso al bus es deteminado por la estación MASTER de una manera exacta. A pesar de la vieja utilización de estos protocolos, aún hoy en día son muy usados en ambientes industriales e inclusive en sistemas que tienen vocación de preservar y proteger vidas humanas, ya que la comunicación está garantizada a nivel de protocolo por no ser de tipo aleatoreo.

Las rutinas de acceso al medio (MAC) se escribieron en TRANSCEIVER.LIB. Básicamente se analiza si existe colisión, si el transceiver está presente o energizado o si en algún momento el transceiver se des-energizó peligrosamente. También se detalla cómo se genera la señal que evita el jabber [8].

DETECCIÓN DE ERRORES

Para la detección de errores se seleccionó el método de cálculo mediante CRC-CCITT (Cyclic Redundancy Code). El CRC-CCITT, captura todos los errores simples y dobles, todos los errores con un número impar de bits, todos los errores de burst de longitud 16 bits o menor, el 99.997% de errores de burst de 17 bits y el 99.998% de errores de burst de 18 bits y mayores.

En este trabajo se desarrolla el método de tablas de "lookup" (crc_table[]) [10], las que facilitan el cálculo rápido por software del CRC. Los detalles se pueden ver en [7] en CRC.LIB y CRC.H. Este método soluciona un problema fundamental. Como usamos para este proyecto el puerto serial, la orientación que tienen las comunicaciones es tipo carácter.

Normalmente los cálculos de CRC se hacen por hardware con registros de desplazamiento de 16 bits y compuertas OR exclusivas, o sea su orientación es al bit.

En este proyecto el CRC está compuesto de dos bytes e internamente se trabaja como un registro CRC de 16 bits, el cual está dividido en un byte llamado CRC de alto orden (8 bits más significativos) y uno CRC de bajo orden (bits menos significativos).

Como resultado del método aplicado, los pasos de la rutina que hace los cálculos de CRC son:

- Hacer XOR del byte de entrada con el byte de

más alto orden del registro CRC para obtener X.

- Hacer un corrimiento de ocho bits hacia la izquierda en el registro CRC.
- Hacer XOR entre el registro CRC con los contenidos de la tabla de "lookup" , usando X como índice.
- Repetir los pasos anteriores para todos los bytes de mensajes.

El detalle del cálculo de X, así como de las 256 constantes de la tabla de "lookup" se pueden ver en [7].

EL OBJETO SIO

En SIODEF.H podemos ver la definición de SIO. En la estructura SIO están todos los apuntadores a los procesos del sistema desde el manejo de el UART y su interface con el transceiver, hasta el nivel de aplicación.

Su definición general para el MASTER es la siguiente (SLAVE tiene un tratamiento similar):

```
typedef struct
   {
   /* DEFINICIONES DE NIVEL MAC*/
   int      uart_base;        /*dirección base de el UART*/
                              .
                              .
   /* apuntadores hacia las rutinas que manejan el UART*/
   BYTE (*readbyte)();   /*apuntador hacia rutina de lectura de
   el UART*/
                              .
                              .
   /*apuntadores hacia las estructuras que manejan el UART*/
   struct vout232_ *dtr    /*apuntador a la estructura DTR*/
                              .
                              .
   /*DEFINICIONES DE NIVEL 2 o de nivel de Protocolo de
   Enlace de Datos*/
   struct protocol pollselect;    /*apuntador a definiciones de
   protocolo*/
   WORD *crc_table;          /*apuntador a la lookup table*/
   BYTE *ring_buff;          /*al buffer en anillo que maneja
   la recepción de datos*/
                              .
                              .
   /*DEFINICIONES DE NIVEL 7 o de nivel de Aplicación*/
   struct video_graf *videog;   /*apuntador a parámetros de
   despliegue en la pantalla*/
   struct terminal *server;                /*apuntador a
   parámetros de terminal desplegada en el servidor*/
   struct terminal *term1;                 /*apuntador a
   parámetros del terminal 1*/
                              .
                              .
   }SIO;
```

El statement typedef nos permite definir nuestro propio objeto de datos, el SIO. Como tal, el posee los atributos de todos los tipos de datos (scope, storage, class, y demás). Esto significa que el compilador lo sujeta a las mismas reglas de chequeo de tipo int. Aquí se ve claramente el uso de un objeto en cuanto a que este proporciona una forma de juntar en una sola unidad las variables y funciones que operan sobre ellas.

Como cualquier objeto, este tiene que ser inicializado. Hagamos el ejemplo para el puerto COMM 1 del PC y definimos:

```
SIO TYPEASIO =
{
/* DEFINICIONES DE NIVEL MAC*/
COMM1,
 .
 .
inportb,
 .
 .
&pin20A,
 .
 .
/*DEFINICIONES DE NIVEL 2 o de nivel de
Protocolo de Enlace de Datos*/
&ps,
Crctable,
SUPPLIED,        /*Significa que este parámetro será
suministrado cuando el programa está corriendo*/
 .
 .
/*DEFINICIONES DE NIVEL 7 o de nivel de Aplicación*/
        &video_default,
        &serv,
        &Ter1,
 .
 .
};
```

Estas definiciones están hechas en los encabezados .h escritos. Por ejemplo COMM1 está en IBMPC.H y tiene la siguiente definición:

```
#define   COMM1   0x3F8
```

Cuando comienza el programa, la siguiente línea inicializa SIO y adicionalmente define el puntero *siop, el cual apunta a SIO:

```
register SIO *siop = &TYPEASIO;
```

Como el lector se ha podido dar cuenta, SIO contiene toda la información del sistema. Pero ¿cómo podemos pasar esta información de una manera eficiente a las rutinas que manejan el sistema?. Para ver la mecánica veamos el siguiente ejemplo.

En BUOS.LIB existe la rutina s_putc () que escribe

un carácter c en el UART especificado en la estructura SIO. La forma más adecuada de pasar SIO en la lista de parámetros de la función, cuando esta es invocada, es pasar un apuntador a la estructura SIO así:
Void s_putc(register SIO *siop, BYTE c).

Internamente esta rutina manejará este apuntador así:

```
void s_putc(register SIO *siop, BYTE c)

{
 while ((*siop->s_xstat)(siop) == NULL); /*esperar que el
UART esté listo*/
 (*siop->s_send)(siop,c);
}.
```

Aquí la rutina s_putc() se apoya en otras, como lo son s_xstat() (estado del UART) y s_send() (enviar carácter al UART), las cuales están definidas en SIO.

SISTEMA DE TEMPORIZACION

Los computadores tienen un metrónomo de 55 ms, el cual se usa para todos los cálculos de temporizadores. Las constantes para este sistema están definidas en TIMER.H y las rutinas necesarias para actuar en conjunto con la máquina de estados se encuentran en TIMER.LIB

Cuando el temporizador arranca (START) se capturan los valores de TIMER_HI y TIMER_LO, dados por el metrónomo del PC, y se calcula el intervalo de tiempo que debe transcurrir para obtener t_tx_result o t_rx_result según el caso puede ser EXP (expiró), UP (activo), TIMER_RES (temporizador en reset). El resultado de este cálculo se guarda en SIO.

MAQUINA DE ESTADOS FINITOS

Una máquina de estados finitos se define como una 4_tupla (S,M,I,T) donde:

S es el conjunto de estados que los procesos pueden tener.
Mes el conjunto de tramas que pueden ser intercambiadas sobre el canal.
I es el conjunto de estados iniciales de los procesos.
T es el conjunto de transiciones entre estados.

En PROTOCOL.H se definen todos los estados (118 en total), estructuras y variables que puede tener MASTER y en PROT_S.H los que puede tener SLAVE por asuntos relacionados con el protocolo. Las variables que tiene el sistema son variables de comando y variables de resultado.

Estas variables son actualizadas al final de la ejecución de un proceso. Las variables están almacenadas en SIO. Cada transición hace una valoración de las variables para obtener el estado del sistema. La rutina que tiene estas transiciones es llamada STATE_RESULT().

El enfoque para cada transición es valorar los estados en la transmisión y en la recepción. Las preguntas que se hacen en el caso de la transmisión son:

¿Cuál fue el último comando de transmisión dado por el protocolo?
¿Cuál fue su resultado al ejecutarlo?
¿Cuál es el estado del temporizador de transmisión?.

Las mismas preguntas se hacen para la recepción. Tras el análisis de las preguntas para transmisión y recepción (seis en total) se puede establecer el estado del proceso.

En el ejemplo que se expondrá, en el cual se evidenciará el estado de "listo para enviar sondeo (poll)" la primera pregunta está almacenada en la variable tx_command, la segunda en tx_result y la tercera en t_tx_result. El estado resultante del proceso se da en la variable process_state:

```
if(...tx_command == TXOFF          &&
  ...tx_result    == NO_PROCESST   &&
  ...t_tx_result  == EXP           &&
  ...rx_command == RXOFF           &&
  ...rx_result    == NO_PROCESSR   &&
  ...t_tx_result   == EXP) {
  ...process_state = STATE1;
  .
  .
  return;
  }
```

Las constantes tienen el siguiente significado:

TXOFF: no se transmita de ninguna trama de protocolo,
NO_PROCESST: no se está procesando ninguna transmisión de protocolo,
EXP: no hay temporización pendiente,
RXOFF: no se reciba de ninguna trama de protocolo,
NO_PROCESSR: no se está procesando ninguna recepción de protocolo,
STATE1: valor dado al estado del sistema tras el

análisis de las variables.

Una vez que se ha definido el estado del sistema, la rutina PROTOCOL_COMMAND() decide según la definición del protocolo que ella tiene almacenado, las acciones a tomar. En el siguiente ejemplo, dado que se puede transmitir el sondeo (STATE1), la rutina PROTOCOL_COMMAND lo ordena en el siguiente segmento:

```
If(...process_state == STATE1) {
  poll_mux(siop);              /*definir cuál es la
estación a la que se deberá dirigir el sondeo*/
  Mk_poll_frame(siop);         /*construir la trama de
sondeo*/
  ...tx_command = TXPOL;       /*transmitir la trama de
sondeo*/
timer(siop, TXTIMER, START) ; /*inicializar el
temporizador de transmisión*/
  ...rx_command = WNAK;        /*estar pendiente si
hay rechazo de la trama de sondeo*/
    timer(siop, RXTIMER, FINISH); /*inhibir el el
temporizador de recepción mientras se está
transmitiendo*/
  return;
}
```

RESULTADOS

La red RS-485 presentó un desempeño aceptable a 115.2 Kbaud. Si bien es cierto que en el experimento los enlaces entre los computadores tenían una longitud de 10 m para un total de 30 m en topología de bus (un MASTER y 3 SLAVE), no se presentaron errores de protocolo. Esto es alentador porque como dicen los expertos "un protocolo es erróneo hasta que se demuestre lo contrario". Sin embargo una prueba del protocolo exigirá una experimentación intensiva.

Las funciones SIO consumen gran cantidad de tiempo de indexamiento relativo al apuntador SIO. El protocolo operó tal de acuerdo a los requerimientos y puede perfeccionarse. El transceiver es presentado en la (Figura 3):

CONCLUSIONES

Se ha mostrado el diseño e implementación de un sistema tele-informático abierto a la experimentación, dada la restricción que tienen los diferentes sistemas implementados por las empresas que producen tanto el hardware como el software.

El sistema es apto para explorar:

Comportamiento de topologías en bus y anillo.
Diferentes tipos de acceso como p-persistente.
Redes auto-curativas.
Modelos de diversa índole como OSI, etc.

Esto es alentador porque como dicen los expertos "un protocolo es erróneo hasta que se demuestre lo contrario". Sin embargo una prueba del protocolo exigirá una experimentación intensiva.

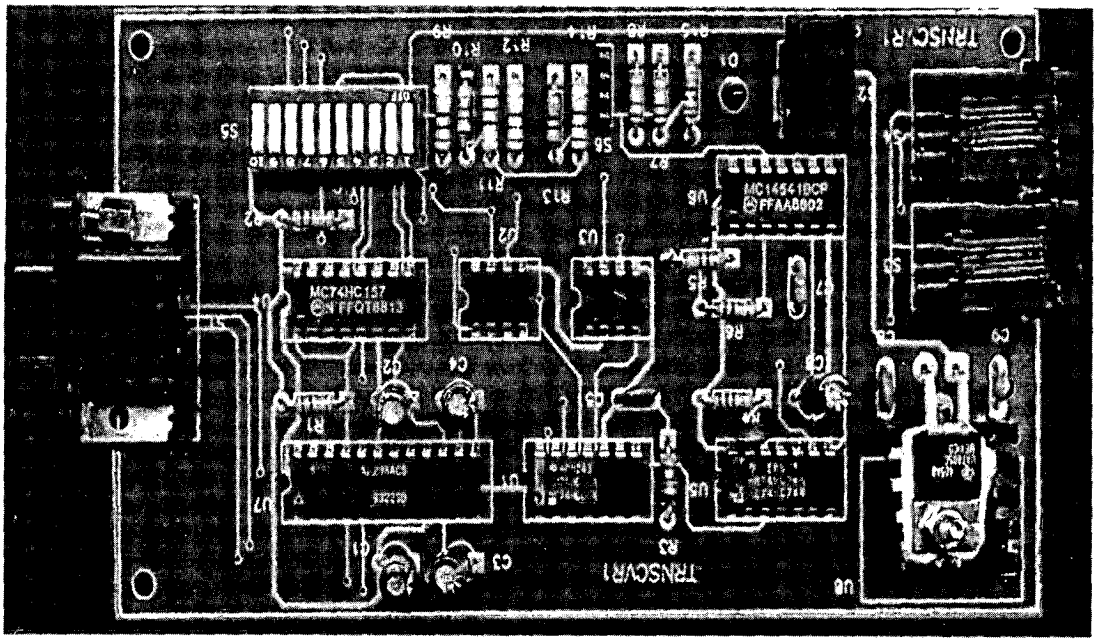

FIGURA 3. Aspecto del Transceiver.

REFERENCIAS

[1] *ELECTRONIC INDUSTRIES ASSOCIATION. Standard for Electrical Charcateristics of Generators and Receivers for use in Balanced Digital Multipoint Systems, EIA_485. Washington, Abril 1993.*

[2] *SCHILDT, Herbert. Turbo C/C++ The Complete Reference. Berkeley: McGraw-Hill, 1.990.*

[3] *FOWLER, B. Transmisión Line Characteristics, National Semiconductor, USA, Mayo 1.974.*

[4] *INTEGRATED CIRCUITS DATA BOOK, Maxim, 1989.*

[5] *NATIONAL SEMICONDUCTOR, Interface Databook. Santa Clara, CA, 1.988.*

[6] *CMOS Logic Data DL131/D Rev3. Motorola Technical Information Center, USA, 1.991.*

[7] *CARDENAS, R. Diseño, Desarrollo e Implementación de una Plataforma Física de red LAN compatible con una red WAN. Universidad Distrital F.J.D.C., Santa Fé de Bogotá, Colombia, 1996.*

[8] *KEISER, G. Local Area Networks, Singapore: McGraw-Hill Book Co, 1989.*

[9] *CUSTER Helen. El Libro de Windows NT. Madrid: McGraw-Hill/Interamericana de México, S.A. 1.994. [31] SCHLAGE ELECTRONICS. Model 708P, Users Manual. USA.*

[10] *CAMPBELL Joe. C Programer's Guide to Serial Communications. (2a. Edición), Indianapolis: Sams Publishing, 1.994.*

Roberto Cardenas C.
Ingeniero Electrónico, U. Distrital
Magíster en Teleinformática, U. Distrital
Profesor de la facultad de Ingeniería en la Maestría en Teleinformática U. Distrital e-mail: bitek@multi.net.co