En el protocolo propuesto permite priorizar los paquetes y proveer al nodo de autonomía, pudiendo decidir si el paquete será retransmitido o tratado localmente. De ésta forma, cada paquete enviado tendrá, como parte de su encabezado, un identificador asociado a su prioridad. Los niveles de prioridad se describen a continuación:

- Paquetes de Nivel 1: Paquetes que contienen datos críticos, cuya pérdida o retraso ocasionan al sistema daños irreparables. Ejemplo: alarmas, ordenes de control críticas.
- Paquetes de Nivel 2: Paquetes que contienen datos críticos, cuya pérdida puede ser tolerada por el sistema. Ejemplo: datos de control para los dispositivos móviles, datos de sensores necesarios para realizar decisiones de control, *beacons* entre vecinos.
- Paquetes de Nivel 3: Paquetes que contienen datos no críticos que, ante una ocasional pérdida requieren retransmisión. Ejemplo: transmisión de audio, vídeo o imágenes.
- Paquetes de Nivel 4: Paquetes que contienen datos que pueden ser procesados en el nodo. Ejemplo: datos para que los nodos realicen tareas locales.

Estas prioridades son consideradas al colocar los paquetes en las colas para su procesamiento dentro del protocolo. Ya sea para su transmisión, reenvío, o cálculo de ruta, los paquetes con mayor prioridad serán procesados primero con el fin de asegurar su llegada sin importar el tráfico de la red.

**Descripción de los Módulos**

La implementación del protocolo se realizó Ada95. Se definió la política de manejo de tareas de igual prioridad, y la política de interbloqueo.

La figura 1 ilustra el módulo de comunicación. El mismo se compone de las siguientes tareas y componentes:

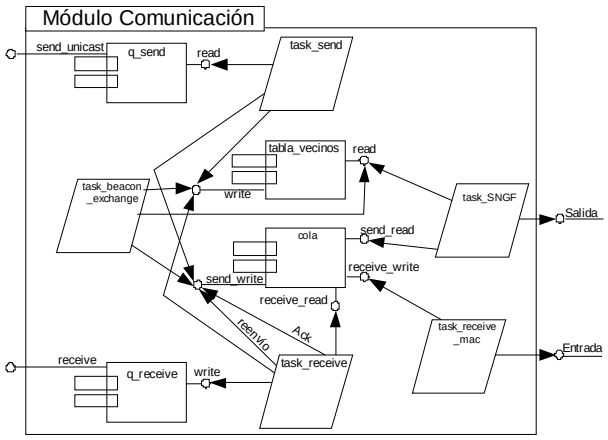


**Figura 1:** Módulo de comunicación

- La tarea *task_send*, lee continuamente de la cola **q_send** en la cual las aplicaciones dejan los mensajes que desean enviar al exterior. Su función principal es dividir los mensajes en paquetes y agregarle a cada uno el encabezado (*header*) para después enviarlo a la cola de paquetes listos (**queue**).
- La tarea *task_receive* lee continuamente la cola de paquetes listos (**queue**) en busca de paquetes de datos provenientes del exterior para que, según su encabezado, armarlo en un mensaje completo y luego pasarlo a la cola **q_receive** de donde las capas superiores los leen. Si alguno de estos paquetes es un ACK (acuse de recibo) en respuesta por un paquete enviado por el nodo, la información contenida por este se utiliza para actualizar la tabla de vecinos (**neighbor_table**). Esta tarea también se encarga de enviar los ACK de cada uno de los paquetes que recibe correctamente, y de reenviar los paquetes entrantes al nodo pero que no están dirigidos a este; en ambos casos los paquetes son enviados a la cola de paquetes listos para su transmisión. Tanto la tarea de *task_send* como *task_receive*, calculan los retrasos en base a la información contenida en los paquetes que reciben, especialmente la de los paquetes ACK provenientes de otros nodos en respuesta a algún paquete enviado, y luego la escriben en la tabla de vecinos.
- La tarea *task_beacon_exchange* se encarga de enviar periódicamente *beacons* de control de retraso a los vecinos (cada un minuto), como de procesar los *beacons* entrantes. Con la información recolectada a través de los datos contenidos en los *beacon* se actualiza la tabla de vecinos.
- La tarea *task_SNGF*, se encarga de decidir, en base a los valores contenidos en la tabla de vecinos, cual de ellos es el más conveniente para enviar el paquete o si su destino se encuentra en el próximo salto a realizar. Si no logra enviar el paquete, enviará un *beacon* de alarma para así anunciar que la ruta está saturada.
- La tarea *task_receive_mac*, lee constantemente de la capa inferior en espera por nuevos paquetes a procesar, estos paquetes los envía directamente a la cola de paquetes listos para las demás tareas los procesen.

Para las estructuras de datos como colas, y tablas que son utilizadas concurrentemente por las tareas, Ada provee mecanismos de bloqueo como los objetos protegidos que simplifican el trabajo de control de concurrencia.

- El componente cola de envío (**q_send**) provee el procedimiento **send_unicast** sirve como interfaz para que las aplicaciones de las capas superiores envíen sus mensajes dirigidos a **addr_dest**, con una prioridad definida por la misma aplicación que desea enviar el mensaje. Estos mensajes son encolados de acuerdo a la prioridad, los de mayor prioridad primero. La lectura de los mensajes de esta cola se realiza mediante la función **read** (cuadro 1).

```
protected q_send is
procedure send_unicast(addr_dest : address;
          p : pack_priority; d: deadline; d : ul_data);
function read : message;
  private
  queue_send : pt_message;
```

**Cuadro 1:** Definición del objeto q_send.

- El componente cola de entrada (**q_receive**) provee la función **receive** que sirve para que las capas superiores lean los mensajes de entrada. El procedimiento **write**, se encarga de escribir en la cola de paquetes recibidos (cuadro 2).

```
protected q_receive is
procedure write(m :message);
function receive : ul_data;
  private
  queue_receive : pt_message;
end q_receive;
```

**Cuadro 2:** Definición del objeto q_receive.

- El componente **neighbor_table** es una tabla de los vecinos detectados junto con sus respectivos valores de retraso de envío, retraso de recibo y otros. Este componente posee dos interfaces: una para lectura de la tabla, y otra a través de la cual se escriben valores en ella a fin de agregar o eliminar un vecino, o actualizar algunos de sus valores (cuadro 3).

```
protected neighbor_table is
procedure read : neighbor;
procedure write (v : neighbor; acc : acction; c : field);
  private
  table : n_table;
end neighbor_table;
```

**Cuadro 3:** Definición del objeto neighbor_table.

- Por último el componente de cola de paquetes listos (**queue**), provee múltiples interfaces de comunicación con las tareas de envío y recibo de paquetes: **send_read**, **send_write**, **receive_read**, **receive_write**. La escritura de *beacons* y paquetes se realizará ordenada de acuerdo a las prioridades definidas en el paquete y, en casos de paquetes de igual prioridad, serán tratados según la política EDF. Por su parte, al leer de cualquier cola, se leerá siempre el primero (cuadro 4).

```
protected queue is
function send_read : package;
procedure send_write(p : package);
function receive_read : package;
procedure receive_write(p : package);
  private
  queue_receive: pt_ package;
  queue_send: pt_package;
procedure insert_priority (queue_dst : pt_package;
          pack : package);
procedure insert_edf (queue_dst : pt_package;
          pack : package, p : pack_priority);
```

**Cuadro 4:** Definición del objeto queue.

La prioridad de los mensajes será definida por la aplicación de capa superior que envía el mensaje, y se mantendrá la misma prioridad para cada uno de los paquetes que compongan tal tarea.

## 3. ARQUITECTURA DEL NODO

En nuestra topología *ad hoc*, los nodos no solamente se encargan de retransmitir los paquetes a sus nodos vecinos, también realizan su planificación siguiendo un criterio preestablecido y tareas de cooperación en el control y monitoreo. La figura 3 ilustra la arquitectura del nodo.

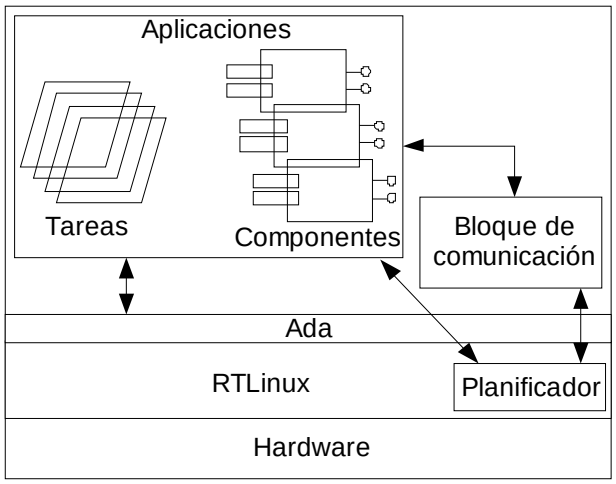


**Figura 2:** Arquitectura del nodo.

El kernel seleccionado para este trabajo fue RT-Linux debido a sus grandes prestaciones en la implementación de STR duros, sumado a su característica de software libre [7]. La portabilidad de Ada sobre RT-Linux fue tomada del proyecto OCERA [8].

La arquitectura planteada en este trabajo propone un estilo arquitectónico orientado a la construcción de aplicaciones y su objetivo fundamental es proporcionar una notación que permita especificar, diseñar, analizar e implementar sistemas de monitoreo y control remoto de tiempo real, con un nivel de abstracción elevado para el usuario final, permitiendo así, reducir la complejidad y el proceso de validación del software desarrollado.

Entre los objetivos específicos esenciales y necesarios para su desarrollo se contemplan los siguientes aspectos:

- Necesidad de integrar modelos de alto nivel de abstracción con las necesidades particulares de los dominios de aplicación, manteniendo la potencialidad de los métodos formales para la verificación y el análisis. Muchos de los errores causados en las fases de especificación y diseño de los sistemas de tiempo real pueden reducirse limitando la posibilidad de cometerlos en estas fases [15].
- Elección del modelo de concurrencia explícita. Este modelo permite tener en cuenta las consideraciones de tiempo real de forma explícita en la fase de diseño y no para la fase de implementación y pruebas como sucede en los modelos de concurrencia implícita. El modelo de concurrencia explícita lleva a definir una serie de elementos necesarios para la comunicación y coordinación entre tareas.

- Reusabilidad del software. Se utilizan librerías de componentes, que pueden ser instanciados para construir sistemas complejos, los cuales a su vez, pueden ser reutilizados en otras aplicaciones.
- Los elementos del estilo arquitectónico propuestos permiten construir software aplicados a los sistemas de monitoreo y control remoto de tiempo real y se clasifican en: actividades, mecanismos de coordinación (comunicación y sincronización), componentes específicos, componentes genéricos o no específicos y componentes pasivos, figura 4. En los trabajos de [14], [16], [10], [6], [5] se presentan una gran cantidad de estos elementos.

## 4. APLICACIÓN

En un área de gran escala, se ubican de forma regular una serie de nodos inalámbricos fijos bajo la topología *ad hoc* y un conjunto de nodos móviles autónomos y semiautónomos se desplazan en ella con trayectorias previamente definidas, figura 3. Cada nodo fijo no solo retransmite información, sino que poseen capacidad de diferenciar entre los datos para ser tratados localmente (corrección de trayectoria de los vehículos, procesamiento de información de control, etc.) o los que requieren retransmisión al nodo más conveniente. A la vez cada uno de los nodos fijos coopera con la estimación de la posición de los nodos móviles mediante triangulación en función de los modelos de radiación WiFi [3].

Este tipo de aplicación encuentra gran utilidad en diversos campos, entre ellos mencionamos el monitoreo de cultivos, de hábitat, ambiental, control de gases nocivos en fábricas, etc. [11], [12], [13].

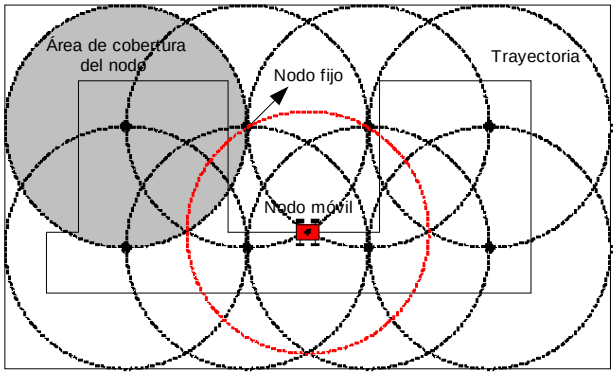


**Figura 3:** Esquema de la aplicación.

**Arquitectura del Nodo de la Aplicación**

La figura 4, muestra la arquitectura del nodo fijo. El Módulo de Control Local se encarga de controlar el movimiento del nodo móvil, para ello realiza el sensado de las variables que en el intervienen. En Status se almacena la información necesaria para poder hacer la estimación de la posición del móvil. El módulo de Monitoreo Ambiental se encarga del control de las variables del tipo temperatura, presión, humedad, etc.; mientras que el de Transmisión de Datos de la adquisición de datos de tipo vídeo, audio y otros.

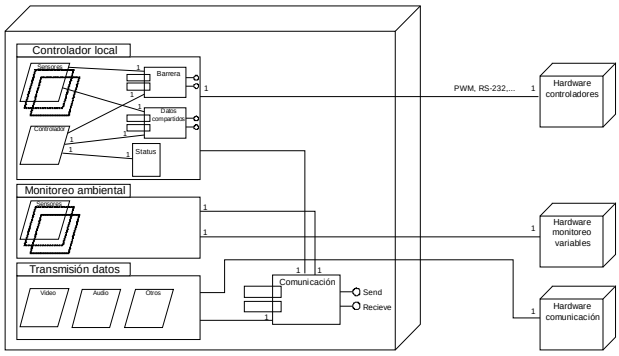


**Figura 4:** Arquitectura del nodo fijo.

**Experiencias**

A fin de validar la arquitectura embebida y las capacidades del protocolo de comunicación para abordar situaciones de tiempo real con datos mixtos, y la cooperación de los nodos fijos en la estimación de la posición de los nodos móviles, se implementó un nodo móvil, el cual se ilustra en la figura 5. El mismo se encuentra equipado con un computador portátil Pentium 133 Mhz, con 16 MB de memoria RAM, tarjeta de red inalámbrica Intel AnyPoint Wireless II Network, sensores ultrasónicos tipo polaroid, cámara web y micrófono.

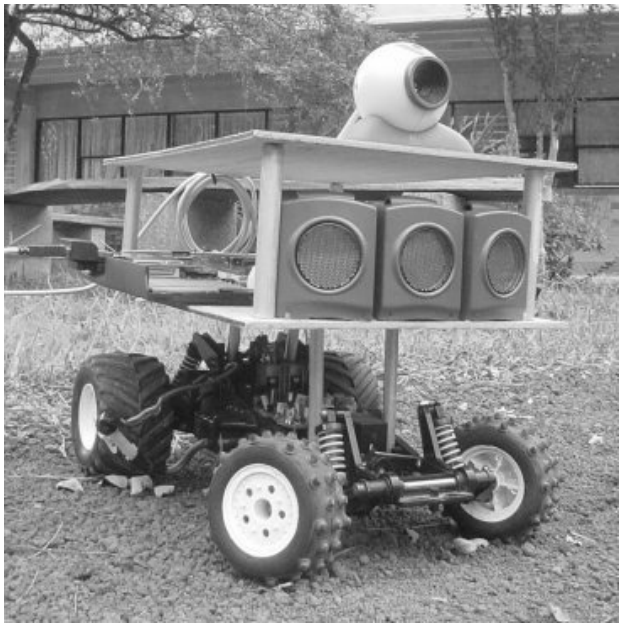

**Figura 5:** Nodo móvil.

Se dispusieron tres nodos fijos implementados con computadores personales con tarjetas de red inalámbrica Intel PRO/Wireless 2011B como se ilustra en la figura 6.

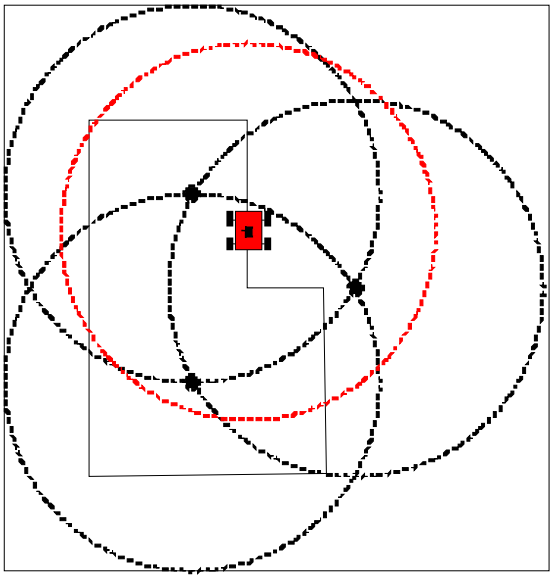

**Figura 5:** Nodo móvil.

## 5. CONCLUSIONES

En este trabajo se muestra una topología y arquitectura de nodos inalámbricos para el control y monitoreo de dispositivos móviles en una red con tráfico mixto, haciendo énfasis en satisfacer los requerimientos de tiempo real de la comunicación.

La topología *ad hoc*, por su independencia de infraestructura, permite cubrir grandes áreas sin la necesidad del costoso cableado. Esto es ideal para proyectos de monitoreo de grandes plantaciones o bosques.

Para abordar la problemática del tráfico mixto fue necesario proveer al protocolo de discriminar entre paquetes críticos y no críticos.

## 6. REFERENCIAS


[1] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, and David Culler, Wireless Sensor Networks for Habitat Monitoring, Intel Research Berkeley, June, 2002.

[2] Per Arne Wiberg and Urban Bilstrup, Wireless technology in industry - applications and user scenarios, in Proc. of the 8:th IEEE International Conference on Emerging Technologies and Factory Automation, Antibes-Juan les Pins, France, October 2001, pp. 123-133.

[3] O. Serrano, J.M. Cañas, V. Matellán, L. Rodero, Robot Localization Using WiFi Signal Network. WAF 2004.

[4] AFT Winfield and OE Holland, The Application of Wireless Local Area Network Technology to the Control of Mobile Robots, Microprocessors and Microsystems, 597-607. 2000.

[5] Martin Torngren. Fundamentals of Implementing Real-Time Control Applications in Distributed Computer Systems. Real Time Systems. Vol 14 No 3. 1998.

[6] Alessandro Orso and Mauro Pezze. An Environment for Designing Real-Time Control Systems. Proceedings of the Firts KIT125 Workshop, Formal Methods for the Design of Real-Time Systems. 1997.

[7] Victor Yodaiken, RT-Linux, White paper, Departament of Computer Science, New Mexico Institute of Technology, avalaibe at http://www.rtlinux.org/documents.

[8] Ismael Ripoll, Alfons Crespo, Adrian Matellanes, Zdenek Hanzalek, Agnes Lanusse and Giuseppe Lipari, OCERA architecture and component definition, OCERA Consortium, 2002. avalaible at http:// www.ocera.org/download/document/repports.

[9] A. Winfield, Distributed Sensing and Data Collection Via Broken Ad-Hoc Wireless Connected Networks of Mobile Robots.In Proceedings of Fifth International Symposium on Distributed Autonomous Robotic Systems (DARS 2000).

[10] Jose L. Fernandez. A Taxonomy of Coordination Mechanisms Used by Real-Time Processes. ACM Ada LETTERS, Volume XVII, Number 2, March/April 1997, Pag 29-54.

[11] Sven Ackmer, Urban Bilstrup y Lisa Svamark. Routing Protocol for Wireless Realtime Multihop Network. Master Thesis, Halmstad University, Halmstad, Sweden. 1999.

[12] Ram Ramanathan y Regina Hain. Topology Control of Multihop Radio Networks using Transmit Power Adjustment. Proc. IEEE Infocom, Tel Aviv, Israel, Marzo 2000.

[13] Hung-Yun Hsieh and Raghupathy Sivakumar. IEEE 802.11 over Multi-hop Wireless Networks: Problems and New Perspectives. IEEE Vehicular Technology Conference (VTC), Vancouver, Canada, Septiembre 2002.

[14] Jose Luis Fernandez. Arquitectura Software Generica para Sistemas de Tiempo Real, Tesis Doctoral, Junio de 1997.

[15] Miguel Felder, Mauro Pezze. A Formal Approach to the Design of Real-Time Systems. Proceedings of the Firts KIT125 Workshop, Formal Methods for the Design of Real-Time Systems. 1997

[16] A. Burns and A. Wellings. RT-HOOD: A Structued Design Method for Real-Time Ada Systems. ELSEVIER. 1995.