UNIVERSIDAD DE CANTABRIA

Facultad de Ciencias

*Proyecto Fin de Carrera*

# DESARROLLO DE UN CONTROLADOR PARA CÁMARAS DIGITALES BASADAS EN EL ESTÁNDAR GIGE VISION

(Development of a driver for digital cameras based on the GigE Vision Standard)

Para acceder al Título de

**INGENIERO EN INFORMÁTICA**

Autor: Pablo Gutiérrez Peón

Diciembre - 2011

FACULTAD DE CIENCIAS

# INGENIERÍA EN INFORMÁTICA

## CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

**Realizado por:** Pablo Gutiérrez Peón
**Director del PFC:** Mario Aldea Rivas
**Título:** "Desarrollo de un controlador para cámaras digitales basadas en el Estándar GigE Vision"
**Title:** "Development of a driver for digital cameras based on the GigE Vision Standard"

**Presentado a examen el día:**

para acceder al Título de

**INGENIERO EN INFORMÁTICA**

Composición del Tribunal:
Presidente: González Harbour, Michael
Secretaria: González Rodríguez, Inés
Vocal: Vallejo Alonso, Fernando
Vocal: Menéndez de Llano Rozas, Rafael
Vocal: Sanz Gil, Roberto

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente Fdo.: La Secretaria

Fdo.: Vocal Fdo.: Vocal

Fdo.: Vocal Fdo.: El Director del PFC

Quiero aprovechar la posibilidad de recordar a esa gente que hace que los esfuerzos valgan la pena.

Gracias a mis padres y a mi hermana, por ayudarme a ser lo que soy.

Gracias a mi abuela y a mis tíos Conchi y José Luis, a Elena, que desde siempre han hecho que tenga en Santander una segunda casa.

A mis amigos Jorge, Arkaitz, Diego, Guti, Toni, David... a todos, también muchas gracias. A Dani, que sobra decir es como un hermano. Ser amigo vuestro ayuda en todo.

Agradecerles a mis compañeros de clase Sandra, Borja, Gerardo, Jose... los buenos momentos.

Y por último, al grupo CTR, sobre todo a Dani y Mario, que a pesar de todas mis dudas me han hecho ver que hay vida en MaRTE.

# Índice de contenidos

**Resumen** ..... **iii**
Summary ..... iv
**1. Introducción** ..... **1**
1.1. Visión por computador ..... 1
1.1.1. Cámaras industriales Ethernet ..... 2
1.2. Entornos a los que se dirige el proyecto ..... 3
1.3. Objetivos del proyecto ..... 4
**2. Antecedentes** ..... **7**
2.1. Sistemas de Tiempo Real ..... 7
2.1.1. Sistema operativo MaRTE OS ..... 9
2.2. Lenguaje de programación Ada ..... 11
2.3. Protocolos para la comunicación de redes: Ethernet, IP, UDP y DHCP ..... 12
2.3.1. Arquitectura de protocolos ..... 12
2.3.2. Ethernet ..... 15
2.3.3. IP ..... 18
2.3.4. UDP ..... 20
2.3.5. DHCP ..... 21
**3. Estándar GigE Vision** ..... **25**
3.1. Introducción y propósito ..... 25
3.1.1. Sistemas GigE Vision ..... 26
3.1.2. Estructuración y funcionamiento ..... 27
3.2. Descubrimiento de dispositivos ..... 28
3.2.1. Fases del descubrimiento ..... 28
3.2.1.1. Configuración IP ..... 28
3.2.1.2. Enumeración de dispositivos ..... 29
3.2.2. Conexión y desconexión del dispositivo ..... 30
3.3. Control del dispositivo (GVCP) ..... 30
3.3.1. El concepto de canal ..... 31
3.3.1.1. Canal de control ..... 31
3.3.1.2. Canal de streaming ..... 34
3.3.1.3. Canal de mensajes ..... 34
3.3.2. Formato de las tramas ..... 35
3.3.2.1. Cabeceras ..... 35
3.3.2.2. Tramas del canal de control ..... 37
3.3.2.3. Tramas del canal de mensajes ..... 40
3.3.3. Fichero XML de configuración del dispositivo ..... 41
3.4. Obtención de imágenes (GVSP) ..... 41

3.4.1. Bloques de datos ..... 42
3.4.2. Formato de las tramas ..... 43
3.4.2.1. Cabecera de datos ..... 43
3.4.2.2. Tramas del canal de streaming ..... 44
3.4.3. Formatos de pixel ..... 45
3.5. Registros ..... 45
3.6. Ejemplo mínimo de utilización de los comandos ..... 46
**4. Implementación ..... 47**
4.1. Visión general ..... 47
4.2. Directrices de implementación ..... 48
4.2.1. Driver Ethernet de MaRTE ..... 48
4.2.2. Modelado propuesto para las tramas ..... 50
4.2.3. Minimización de copias de datos ..... 57
4.2.4. Herramienta de análisis de protocolos Wireshark ..... 61
4.3. Implementación de los protocolos ..... 63
4.3.1. Ethernet, IP y UDP ..... 63
4.3.2. DHCP ..... 65
4.3.3. GVCP ..... 66
4.3.3.1. GVCP Command ..... 66
4.3.3.2. GVCP Acknowledge ..... 68
4.3.4. GVSP ..... 69
4.4. Implementación de paquetes que hacen uso de GVCP y GVSP ..... 71
4.4.1. Paquete GVCP_Operations ..... 71
4.4.2. Paquete GVSP_Operations ..... 72
4.4.3. Paquete Gige_Camera ..... 74
4.5. Patrones de uso ..... 76
**5. Evaluación y pruebas ..... 77**
5.1. Medida de prestaciones ..... 78
**6. Conclusiones y trabajo futuro ..... 83**
6.1. Conclusiones ..... 83
6.2. Trabajo futuro ..... 84
**Bibliografía ..... 85**
**Glosario ..... 87**

# Resumen

Los sistemas de visión artificial por computador constituyen una de las herramientas más potentes para obtener información del entorno que nos rodea. Si se consiguen interpretar correctamente los datos obtenidos mediante estas plataformas, se podrán tomar decisiones fiables en base a ellos que permitirán desplegar un amplio abanico de aplicaciones en campos como la robótica, medicina, cartografía, tráfico, agricultura o control de calidad entre otros muchos.

Uno de los sectores que ha aprovechado más las potenciales posibilidades que estos sistemas ofrecían para resolver problemas comunes en su ámbito de trabajo es el sector industrial.

Este Proyecto Fin de Carrera se ha fijado como objetivo el desarrollo de un controlador para una cámara digital de características industriales (fiabilidad, resistencia, rapidez, etc.) que pueda ser empleado en aplicaciones industriales de análisis y control en tiempo real.

Dentro del mundo de las cámaras de video digitales industriales, existe un tipo que resulta muy interesante debido a la interfaz de conexión con la que cuentan para enviar los datos: Ethernet. La conocida como Automated Imaging Association (AIA), agrupa a las empresas más importantes de este sector y ha desarrollado un estándar para la transmisión de vídeo y control de dispositivos sobre Ethernet denominado GigE Vision.

Con la definición de este estándar, se da una interfaz única para todos los dispositivos que la cumplan. Esto permite, entre otras ventajas, emplear cámaras de distintos fabricantes sin necesidad de modificar las infraestructuras y aplicaciones.

El controlador para cámaras GigE Vision creado en el proyecto ha sido implementado para el sistema operativo de tiempo real MaRTE OS, desarrollado en el Grupo de Computadores y Tiempo Real de la Universidad de Cantabria.

Dada la extensión del estándar, no se han implementado todas las funcionalidades en él descritas, aunque en todo momento se pensó en programar el controlador de forma modular para que no resultara complicado añadirlas en un futuro si fuera necesario. El sistema objetivo del diseño fue aquel en el que un computador corriendo MaRTE OS y una cámara se encontraran conectados en la misma red y se permitiera al usuario mediante sencillas llamadas inicializar el dispositivo y capturar imágenes.

Como resultado adicional y debido a que era necesaria una base sobre la que construir los protocolos que define el estándar GigE Vision, se ha implementado en MaRTE OS la pila de protocolos formada por Ethernet, IP y UDP. Así mismo, debido a que el dispositivo precisa que se le otorgue una dirección IP para poder hacer uso de él, se ha implementado un servidor DHCP mínimo que cumpla esta funcionalidad.

# Summary

Computer vision systems are one of the most powerful tools to obtain information from the surrounding environment. If the data obtained by this kind of platforms are correct, we could make reliable decisions based on them which would allow us to develop a wide range of applications in such different fields as robotic, medicine, cartography, traffic, agriculture or quality control among many others.

One of the economic sectors that has taken better advantage of the features offered by these computer vision systems to solve their common problems is the industrial one.

This End of Degree Project has had the purpose of developing a driver for a digital camera which has industrial features like reliability, strength and speed. Besides, this driver has been designed to be used in a real-time operating system to allow creating analysis and control real-time applications.

There is a type of industrial digital video cameras which is very interesting because of the connection interface they use to send data: Ethernet. The Automated Imaging Association (AIA) brings together the most important enterprises in this application field and has developed a video streaming and device control over Ethernet standard called GigE Vision.

The standard gives a common unique connection interface to any devices developed under it. This allows, for example, to use cameras from different manufacturers with no change in the networks and applications.

The GigE Vision camera controller created in this project has been developed for MaRTE OS, a real-time operating system developed by the Computers and Real Time Group from the University of Cantabria.

Because of the size of the standard, which covers a wide range of devices and situations, some of the features have not been developed. However, since the project has been created using a modular scheme, new extensions could be easily added in the future if needed. The kind of system in which this project is focused is the one in which there is a computer running MaRTE OS, a GigE Vision camera and both are connected in the same subnet. A set of simple calls are provided to allow the user to initialize and capture images from a camera in a straightforward way.

It was necessary to give a base from which the GigE Vision standard protocols could be built so, as an additional result, a new communication interface using Ethernet, IP and UDP protocol stack was created for MaRTE. Additionally, a minimal DHCP server was developed in order to allow a device to configure its IP address, which is a compulsory step to allow an application to use a camera.

# 1. Introducción

## 1.1. Visión por computador

Los seres humanos percibimos la información necesaria para interactuar con el mundo que nos rodea mediante los sentidos. De los cinco existentes, la vista se destaca como el más importante y complejo de todos. Basta un ejemplo para constatarlo: existen 30.000 fibras nerviosas para el sentido auditivo, mientras que el visual hace uso de 2.000.000 (Escalera 2001). No es de extrañar por tanto que algunos estudios cifren la información generada por la visión y procesada por el cerebro en el 75% de la información total procedente de los sentidos.

Los sentidos producen sensaciones que en unión con experiencias vividas anteriormente crean percepciones. La percepción capta la realidad exterior, la ordena coherentemente y extrae una información de la que nosotros hacemos uso. En base a esta información, podemos distinguir las cosas que se mueven y las que no, calcular trayectorias, reconocer personas, analizar el color de las imágenes, etc.

Esta pequeña introducción sirve como analogía entre la visión humana y la que se tratará en este proyecto: la visión por computador. Se podría definir esta última como el campo de la informática que se encarga del procesamiento automático de imágenes procedentes del mundo real para extraer e interpretar la información que contienen.

Tanto la visión humana como la visión por computador se parecen en sus objetivos: interpretar información para tomar decisiones en base a ella. También se puede decir que tanto el ojo como la cámara tienen características comunes: elementos sensores (retina y sensores CCD por ejemplo), el iris, la córnea y el cristalino serían en la cámara su lente, etc. Por otro lado, el cerebro y el computador serían los elementos de procesamiento. Tanto el cerebro como el computador se caracterizan por manejar la información en base a señales eléctricas.

Los computadores tienen como sentidos a sus dispositivos de entrada. Desde el típico teclado a los sensores que podemos conectar para detectar distintas magnitudes físicas o químicas como la temperatura, distancia, aceleración, humedad, etc.

Si el 75% de la información sensorial que recibe el cerebro procede de la vista, es lógico pensar que no emplear la imagen como medio para obtener datos del exterior en un sistema electrónico supondría un desperdicio y una limitación, ya que no siempre puede sustituirse el trabajo que realiza una cámara por el de otros sensores.

Llegados a este punto, se puede afirmar que la visión por computador está ampliamente justificada. De las tecnologías a las cuales la visión por computador se aplica, dos son las que destacan por encima de otras:

- **Procesamiento de imágenes:** Transforman una imagen en otra con el objetivo de mejorar su calidad y/o facilitar la búsqueda de información.
- **Reconocimiento de patrones:** Clasifican objetos en función de unas características dadas. Se realiza de la mano de los conocimientos de inteligencia artificial.

Cada vez son más ejemplos de aplicaciones que han dado cuenta de lo que la visión por computador puede ofrecer. Estos son algunos, aunque hay infinidad de ellos:

- **Control de calidad:** Se aplica para realizar la inspección de piezas fabricadas y detectar posibles defectos.
- **Agricultura:** El análisis de imágenes tomadas con satélite permite determinar los diversos tipos de terreno y ajustar los usos.
- **Identificación:** Mediante el análisis de las facciones de la cara o de las huellas dactilares.
- **Tráfico:** Desde el ya habitual en muchos aparcamientos reconocimiento de matriculas hasta los coches que se conducen sin piloto.
- **Biomedicina:** La primera aplicación en este campo fue la determinación automática del número de glóbulos rojos en la sangre. Hoy en día se elaboran modelos tridimensionales a través de las imágenes obtenidas por resonancia magnética.
- **Robótica:** Los robots industriales empleados en cadenas de montaje y empaquetado deben tener una gran precisión en la colocación de las piezas u objetos, sabiendo en cada momento que hay en el entorno que les rodea.

### 1.1.1. Cámaras industriales Ethernet

Dentro de un sistema de visión por computador el elemento más relevante es la cámara. Son muchos los tipos de cámaras que existen dependiendo de las prestaciones que se deseen. La primera distinción surge entre *cámaras fotográficas* y *cámaras de vídeo*. Son estas últimas las que aportan imágenes de forma continuada y las adecuadas para los propósitos de este proyecto. A su vez, dentro de las cámaras de vídeo surgen muchos ejemplos: cámaras de cine y televisión, cámaras domésticas, cámaras web, etc.

Las que centran el interés del proyecto son las **cámaras industriales**, es decir, aquellas que han sido creadas para satisfacer las demandas de los procesos de la industria. Características comunes de este tipo de cámaras son su velocidad, fiabilidad o resistencia a agresiones externas.

La evolución sufrida por las cámaras industriales ha ido de la mano de la del mundo de las cámaras en general. Un paso fundamental lo supuso el cambio de cámaras analógicas a **cámaras digitales**. Esta evolución permite disfrutar de las ventajas de la señal digital:

- Cuando una señal digital es atenuada o experimenta leves perturbaciones, puede ser **reconstruida** sin amplificación de ruido y errores como ocurre con la señal analógica.
- Se pueden implementar sistemas de **detección y corrección de errores** sobre los datos digitales.
- **Facilidad de procesamiento**. Se permite una conexión directa con el computador sin necesidad de conversores de analógico a digital.
- Se pueden aplicar técnicas de **compresión de datos** sin pérdidas o con pérdidas, pero mucho más eficientes que con señales analógicas.
- Permite **mayor frecuencia de captura** de imágenes.

Este cambio propicia también la posibilidad de adoptar nuevas interfaces para la conexión con la cámara. Una de las interfaces más empleadas para interconexión de computadores y dispositivos y que aun gozando de gran éxito sigue creciendo en adeptos es **Ethernet**, la

conocida familia de tecnologías para redes de ordenadores de área local. En el Apartado 2.3.2, "Ethernet" se tratará con más detalle sus características, pero a modo de resumen y para lo que se pretende indicar aquí, comentar que Ethernet proporciona **capacidad de interconexión, sencillez y velocidad**.

Por su parte, existe un grupo de cámaras digitales que emplean la interfaz Ethernet y que se rigen por un nuevo estándar denominado **GigE Vision**. Este estándar define para los dispositivos que lo soporten la forma en que pueden ser controlados y cómo deben transmitir video entre ellos.

Permite a su vez un método de conexión sencillo y de gran velocidad entre dispositivos GigE Vision y tarjetas de red empleando medios físicos de conexión Ethernet que son de bajo coste y permiten transmisión de calidad y velocidad a largas distancias.

A juzgar por la gran cantidad de productos basados en esta tecnología que las empresas especializadas en cámaras industriales ofertan en sus sitios web y el apoyo de las principales asociaciones de visión tanto europeas como americanas, esta tecnología está teniendo bastante buena acogida.

## 1.2. Entornos a los que se dirige el proyecto

El propósito de este proyecto es crear un controlador para dispositivos basados en el estándar GigE Vision que pueda ser empleado en el sistema operativo de tiempo real MaRTE OS con el fin de utilizarlo en aplicaciones de control de tiempo real.

Actualmente, MaRTE se utiliza en entornos de tipo industrial en que es común manejar varios computadores conectados por un bus (ver Figura 1.1) y a su vez los computadores se emplean para el control de robots (brazos robóticos, pequeños vehículos, etc.).

Se pretende crear un controlador que permita desde MaRTE manejar cámaras GigE Vision cuya interfaz de conexión es Ethernet, de forma que las capturas realizadas puedan ser empleadas para el manejo de los robots descritos.

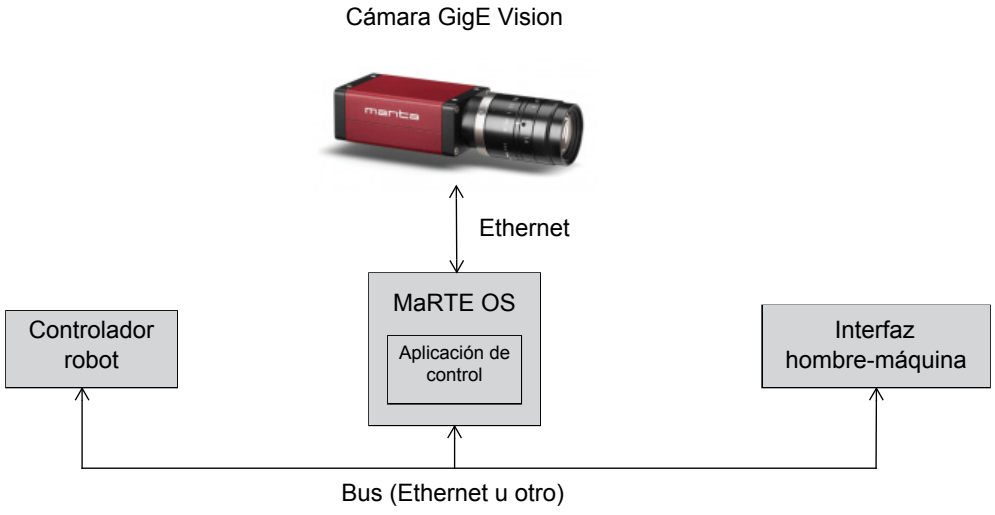


**Figura 1.1: Ejemplo de entorno en que se empleará el driver.**

En muchas ocasiones, en estos entornos se imponen **requisitos a la captura de imágenes** por cuestiones técnicas de diversos tipos. GigE Vision da la posibilidad de controlar estos parámetros. Algunos de ellos son:

- **Formato de la imagen:** Escala de grises o color. Profundidad de cada pixel.
- **Velocidad de captura:** Dependiendo del número de imágenes que se desee obtener.
- **Región de Interés (ROI) o ventana de captura:** A veces no es necesario obtener una captura al máximo de resolución que puede entregar la cámara porque no interesa toda la zona de visualización cubierta. Con ROI se permite realizar una captura de una zona concreta de la imagen.

# 1.3. Objetivos del proyecto

Este proyecto pretende dotar al **sistema operativo de tiempo real** MaRTE OS de las capacidades de adquisición de imágenes que las novedosas cámaras que emplean el estándar GigE Vision otorgan.

El proyecto viene motivado por la existencia en MaRTE OS de un controlador para tarjetas BTTV que permiten la conversión a señal digital de la señal analógica dada por una cámara analógica. Este sistema resultaba a todas luces muy limitado para el campo de aplicación que se perseguía: aplicaciones de control en tiempo real. El número de imágenes obtenido resultaba muy inferior al que potencialmente ofrecen los nuevos dispositivos digitales, los cuales disponen de ventajosas características como la selección de la ventana de captura. La frecuencia de captura es un factor que resulta muy importante si nos movemos dentro del terreno de las aplicaciones de control en tiempo real basadas en análisis de imagen.

Analizando el mercado de cámaras digitales, se encontraron aquellas basadas en el estándar GigE Vision, que por su sencillez de conexión al computador al emplear Ethernet resultan más que interesantes. Se dispone a su vez de un driver Ethernet en MaRTE. Por ello, surge esta idea de proyecto que pretende **dotar el sistema MaRTE de un controlador** que permita una utilización básica de este tipo de cámaras.

El estándar GigE Vision se definió para aglutinar bajo el paraguas del objetivo de la transmisión de video la experiencia obtenida anteriormente con gran cantidad de dispositivos de imagen y tecnologías existentes. Por lo tanto, sus definiciones son amplias con objeto de poder ser amoldables al entorno que se desee modelar. En consecuencia, el hecho de que en este proyecto se haya pretendido adoptar el estándar no implica su total implementación, puesto que cuenta con gran cantidad de características opcionales y recomendadas que por tanto no son obligatorias y que deben tenerse en cuenta en función del campo de aplicación.

De acuerdo con estos objetivos generales, se derivaron los siguientes más concretos:

- Creación de un controlador para una cámara GigE Vision:
  - Implementando parcialmente el estándar GigE Vision.
  - Modular para facilitar su uso y ampliación futura.
  - Midiendo los tiempos de ejecución de los servicios implementados, de modo que puedan utilizarse para realizar análisis de la respuesta temporal de las aplicaciones de tiempo real que los utilicen.

- Creación de la pila de protocolos Ethernet, IP (*Internet Protocol*) y UDP (*User Datagram Protocol*) sobre la que se apoya GigE Vision:
  - Resultado independiente del proyecto que puede ser empleado en otros ámbitos.

- Creación de un servidor DHCP (*Dynamic Host Configuration Protocol*) mínimo empleado en GigE Vision:
  - Para realizar negociaciones empleando siempre los mismos parámetros básicos.
  - Ampliable para poder aumentar su capacidad de respuesta.

Una explicación más detallada de estos objetivos concretos es la siguiente:

En este proyecto se optó por definir de forma clara desde un principio un conjunto de requisitos básicos que permitieran desarrollar un entorno como el descrito en el Apartado 1.2, "Entornos a los que se dirige el proyecto", en donde se cuenta con un sistema básico que cumple con GigE Vision. Una vez este sistema funcionara satisfactoriamente, el objetivo planteado habría sido cubierto. No obstante, en todo momento se ha desarrollado pensado en ampliaciones futuras si fueran necesarias, con lo que la **modularidad** se ha situado como un objetivo más.

El estándar cubre muchos y variados aspectos que permiten implementaciones diferentes en función de las necesidades. Una de las características más condicionantes a la hora de implementar es la topología de la red Ethernet, pudiendo ser esta más o menos compleja. En el proyecto se optó por conectar el controlador y el dispositivo con topología punto a punto mediante un cable cruzado (Figura 1.2).

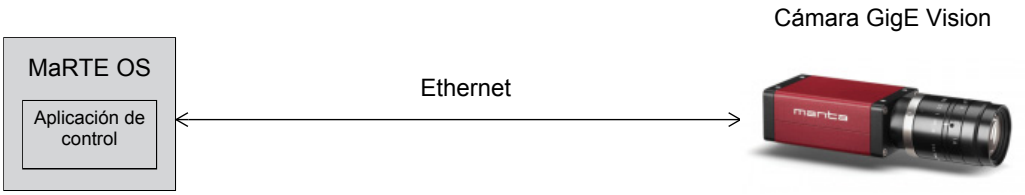


**Figura 1.2: Sistema a implementar propuesto, con topología punto a punto.**

Mediante este esquema se simplifican muchas de las tareas necesarias para que el objetivo final de captura de imágenes sea cubierto. Por ejemplo, no hay que contemplar interacciones inesperadas de otros dispositivos o dar control sobre el dispositivo a varias aplicaciones, entre otras.

Otro de los objetivos siempre presentes en el proyecto era conseguir que todo el **procesamiento necesario fuera suficientemente rápido** como para que el cambio del sistema analógico a digital hubiese merecido la pena. Dadas las especificaciones del fabricante, que ofrecen una tasa de capturas mucho mayor a la que se consigue con cámaras analógicas, el aspecto hardware de la cámara está cubierto. Por tanto, el reto se traslada al desarrollo del controlador en MaRTE OS. Es en este punto donde se encontrará el principal cuello de botella.

Adicionalmente se vio necesario **crear una pila de protocolos de comunicación para MaRTE OS formada por Ethernet, IP y UDP** que permitiera enviar y recibir tramas de estos tipos y transmitir a su vez las nuevas tramas del estándar GigE Vision, ya que corren sobre UDP. También tras el estudio del estándar se percibió la necesidad de **crear un servidor DHCP** mínimo que permitiera dar una dirección IP a la cámara con la cual poder trabajar.

# 2. Antecedentes

## 2.1. Sistemas de Tiempo Real

Un sistema de tiempo real es una combinación de uno o más computadores, dispositivos hardware de entrada/salida y software de propósito específico (González y Palencia 2009) en el cual:

- existe una fuerte interacción con el entorno (en este proyecto mediante la cámara),
- el entorno cambia con el tiempo (la imagen varía con el tiempo),
- el sistema simultáneamente controla y/o reacciona a diferentes aspectos del entorno (se toman decisiones en función de la imagen obtenida para controlar, por ejemplo, robots).

Como resultado:

- los requerimientos temporales están impuestos por el propio software,
- el software es naturalmente concurrente.

Para asegurar que los requisitos temporales se cumplen, el comportamiento del sistema debe ser **predecible**.

En los sistemas de tiempo real la ejecución de cada actividad es disparada mediante la llegada de estímulos llamados *eventos*. Ejemplos de eventos son aquellos producidos por un reloj que pone a ejecutar actividades periódicas, una petición de interrupción procedente de un dispositivo hardware, etc.

La propia actividad que ha sido accionada se denomina respuesta. Habitualmente, cada respuesta es implementada por un *hilo* de control independiente.

La ejecución de estas respuestas puede tener requerimientos temporales que deben ser tenidos en cuenta. En la mayor parte de los casos, estos requerimientos temporales se refieren al máximo tiempo de respuesta que se permite desde la llegada del evento que la activó. Este tipo de requerimiento se denomina *plazo*.

En aplicaciones de tiempo real, la corrección de la computación realizada depende no sólo de los resultados, sino del *instante* en que estos han sido generados. Por ejemplo, no sería válido un estímulo enviado a un robot si no se han podido interpretar los datos de la cámara en un tiempo prudencial de respuesta.

Los sistemas de tiempo real normalmente consisten en hilos independientes de control denominados *tareas*. El término tarea se refiere a cualquier unidad de procesamiento que sea planificable. Este perfil se puede aplicar, por ejemplo, a los procesos y threads con los que es común estar más familiarizado si se trabaja sobre sistemas operativos o en determinados lenguajes de programación. En una posible aplicación del driver desarrollado en este proyecto, podría asignarse una tarea para capturar imágenes y otra para interpretar los datos y devolver los resultados.

Las tareas pueden ser periódicas o aperiódicas:

- Las **tareas periódicas** se activan a intervalos regulares.
- Las **tareas aperiódicas** se activan a intervalos de tiempo irregulares.

Además, las tareas tienen deadlines que pueden ser estrictos o no estrictos:

- Fallar al cumplir los **deadlines estrictos** (*hard deadlines*) da como resultado un fallo del sistema.
- Fallar al cumplir los **deadlines no estrictos** (*soft deadlines*) degrada el rendimiento del sistema.

Las tareas también pueden ser clasificadas como **críticas** y **no críticas**. El fallo de una tarea crítica al cumplir su deadline (si existiera), se considera catastrófico.

En definitiva, los requisitos de un sistema de tiempo real deben ser:

- **Permitir la ejecución de tareas periódicas**, completándolas antes de su deadline.
- **Responder a eventos aperiódicos** que en algunos casos también tienen deadline. Comprometernos a alcanzar un tiempo medio de respuesta preestablecido.
- Asegurar que **cumplimos los deadlines críticos**, incluso en periodos de sobrecarga.
- Asegurar la **consistencia de los recursos compartidos**.

Dado que la cámara que se maneja permite flujos de imágenes continuos a la velocidad que especifique el usuario, podemos afirmar que su tiempo de respuesta está acotado y es proporcional al tamaño de los datos solicitados. De la misma forma, el tiempo de procesamiento realizado por la aplicación para capturar la imagen debe estar acotado también para poder predecir el comportamiento del sistema.

Diferencias entre los sistemas de tiempo real y los sistemas convencionales

En los sistemas de tiempo compartido convencionales, los objetivos que se pretenden son un alto rendimiento, rápida velocidad de respuesta y equidad en el reparto de los recursos entre los usuarios.

Objetivos sustancialmente diferentes son los que se buscan en los sistemas de tiempo real. En particular, la programación en tiempo real persigue mejorar la rapidez de respuesta media, mientras que en los sistemas de tiempo real, se busca que el comportamiento en la peor situación posible sea admisible.

En estos sistemas, el cumplimiento de los deadlines es extremadamente importante. La *planificabilidad* es la herrramienta que nos permite conocer si los deadlines se cumplen. Podríamos interpretar la planificabilidad como la herramienta que permite, si se cogiera un sistema convencional, descartar aquellas tareas que no van a poder cumplir con sus deadlines.

Una visión comparativa entre ambos tipos de sistemas es la que se da en la Tabla 2.1.

**Tabla 2.1: Comparativa entre sistemas operativos de tiempo compartido y de tiempo real.**

| | Sistemas operativos de tiempo compartido | Sistemas operativos de tiempo real |
|---|---|---|
| **Capacidad** | Potencia de cálculo | Habilidad para cumplir los requerimientos temporales: planificabilidad |
| **Respuesta temporal** | Rápida respuesta en promedio | Respuesta de peor caso garantizada |
| **Sobrecarga** | Reparto equitativo | Estabilidad |

### 2.1.1. Sistema operativo MaRTE OS

MaRTE OS (*Minimal Real Time Operating System for Embedded Applications*) es un sistema de tiempo real estricto pensado para **aplicaciones embebidas** que cumple el perfil de sistema de tiempo real mínimo definido en el estándar POSIX.13 (MaRTE OS 2011). Está desarrollado por el *Grupo de Computadores y Tiempo Real* de la *Universidad de Cantabria*. Provee un entorno de fácil uso para desarrollar aplicaciones de tiempo real multi thread. Algunas de las características de MaRTE OS son las siguientes:

- Cumple con el **perfil mínimo de POSIX.13**. Cuenta por tanto con pthreads, mutexes, variables condicionales, etc.
- Implementa el **anexo de tiempo real de Ada 2005**.
- Todos los servicios tienen **tiempos de respuesta acotados**.
- Cuenta con un **único espacio de direcciones compartido** tanto por el núcleo como por la aplicación.
- Núcleo **monolítico**.
- **Escrito en Ada**, con pequeñas partes en C y lenguaje ensamblador.
- Permite **ejecutar aplicaciones concurrentes de C, C++ y Ada**.
- **Portable** a diferentes plataformas (x86, MC68332 y Linux).
- Es **código libre** bajo licencia GNU GPL 2.

Además, MaRTE permite ser compilado para tres arquitecturas diferentes, por lo que podremos disponer de distintos entornos de desarrollo y pruebas:

- **x86:** Permite ejecutar MaRTE sobre un equipo externo (o emulado) con arquitectura x86.
- **linux:** Permite ejecutar MaRTE como un proceso Linux.
- **linux_lib:** Permite ejecutar MaRTE como un proceso Linux con acceso a las librerías GLIBC del sistema.

Utilización de MaRTE en el proyecto

Como se ha comentado en el Apartado 1.3, “Objetivos del proyecto”, el propósito perseguido es desarrollar un driver para el sistema operativo MaRTE que permita aprovechar las capacidades de las cámaras GigE Visión disponibles en el mercado. En este contexto, MaRTE se compiló con arquitectura x86 ya que se quiere ejecutar en un equipo externo. De los lenguajes que ofrece el sistema, se optó por utilizar Ada (ver Apartado 2.2, “Lenguaje de programación

Ada"). No se han empleado las características de tiempo real que define el estándar POSIX.13 ni el anexo de tiempo real del lenguaje Ada. Esto es debido a que el código desarrollado es de muy bajo nivel y en ese contexto no están disponibles estos servicios.

Debido a que los sistemas operativos de tiempo real suelen utilizarse en aplicaciones empotradas, lo habitual es emplear un entorno de desarrollo cruzado en el cual contamos con dos partes:

- **Plataforma de desarrollo** (*host*): Equipo que soporte las herramientas necesarias para editar, compilar y enlazar. En este caso, equipo Linux con el compilador GNAT y el entorno de desarrollo GPS.
- **Plataforma de ejecución** (*target*): Equipo que ejecuta la aplicación. El destino final más habitual es un equipo empotrado. Para este proyecto se ha empleado un PC con procesador Pentium III a 800 Mhz y 256 MB de memoria RAM. Mediante una pantalla se observa la salida del programa.

Por otro lado, en este proyecto se cuenta también con un elemento más: la cámara GigE Vision. Los tres componentes se han conectado entre sí empleando un *puente* Ethernet. De esta forma, los tres tienen visibilidad entre sí en el caso de que tuvieran que enviarse tramas. No obstante, en el entorno final, la única conexión exigible es la existente entre la plataforma de ejecución y la cámara, pues una vez el ejecutable se encuentra en el equipo de ejecución, la plataforma de desarrollo ya no es necesaria.

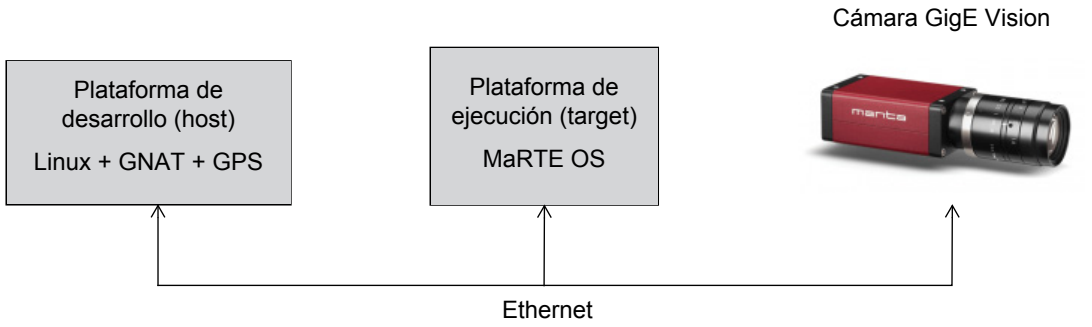


**Figura 2.1: Entorno de desarrollo.**

El ciclo que se ha repetido a lo largo del desarrollo ha sido el siguiente:

1. Se desarrolla la aplicación en el equipo Linux, se compila con GNAT y se enlaza con las librerías de MaRTE generándose el programa ejecutable.
2. Se copia dicho programa a una carpeta compartida con el computador de ejecución. El cargador del computador de ejecución descarga el programa mediante la conexión Ethernet y lo lanza.
3. Si el programa requiere interacción con la cámara, esta debe estar conectada al puente.
4. Los resultados de ejecución se pueden ver en la pantalla del target.

En el estándar Ethernet, los mensajes se entregan a todos los nodos de la red. Sin embargo, si no son de difusión, los nodos que no sean destino del mensaje lo descartan.

## 2.2. Lenguaje de programación Ada

El sistema operativo MaRTE da la posibilidad de programar en alto nivel en dos lenguajes diferentes: Ada y C. Como es de esperar, ambos cuentan con sus ventajas e inconvenientes. El hecho de que el propio sistema operativo, entre cuyos objetivos por ser de tiempo real se encuentran la fiabilidad y robustez, haya descartado otros lenguajes en favor de estos es en sí un motivo más a tener en cuenta para la elección propia.

**C** tiene una ganada reputación como lenguaje base para la implementación de sistemas operativos. A pesar de ser un lenguaje de alto nivel, al poseer estructuras típicas de estos, también cuenta con construcciones de bajo nivel que permiten al compilador mezclar código C con ensamblador o acceder directamente a memoria y dispositivos periféricos. De aquí se deduce su importancia en la implementación de SO.

Por otro lado se tiene al lenguaje de programación **Ada**. A pesar de que su uso no está tan ampliamente extendido como ocurre con el propio C o con Java, Ada es el lenguaje dominante en dos áreas: por un lado los **sistemas grandes, complejos y de larga vida** y por otro los **sistemas críticos de seguridad y los de tiempo real** empleados por ejemplo en aviones o automóviles (Barnes 2005). Estos dominios son aquellos en los que un fallo del software pondría la vida humana en peligro y en los que el dominio de Ada se justifica con sus cualidades.

A modo de nota histórica, Ada fue desarrollado por el Departamento de Defensa de EEUU y su nombre es un homenaje a Lady Augusta *Ada* Byron, considerada el primer programador de la historia.

El lenguaje de programación finalmente escogido para programar íntegramente este proyecto ha sido Ada.

Su elección se ha basado en las siguientes cualidades (González, Gutiérrez y Pérez 2009):

- **Fiabilidad:**
  - **Legibilidad:** Su sintaxis es bastante legible incluso para personas que no estén familiarizadas con el lenguaje. No se escatima en la longitud de las palabras clave y tampoco se recomienda hacerlo en el nombrado de variables. Esta filosofía se basa en que *un programa se escribe una vez, pero se modifica y se lee muchas más veces*.
  - **Tipificación estricta:** El lenguaje obliga a no mezclar datos de distintos tipos en expresiones, y así se detectan más errores de manera automática.
  - **Excepciones:** Permite el tratamiento de errores. De esta forma, estos pasan menos inadvertidos.
- **Modularidad:** El programa se divide en trozos o módulos independientes que se pueden manejar más fácilmente al encontrarse separados.
- **Abstracción de datos y tipos:** Capacidad de hacer módulos independientes de un determinado tipo de datos.
- **Compilación separada:** Capacidad de compilar por separado los diferentes módulos del programa.
- **Concurrencia:** Capacidad para ejecutar a la vez varias partes del programa, que atienden de manera simultánea a varios subsistemas o funcionalidades.
- **Tiempo Real:** Capacidad para garantizar que los tiempos de respuesta del programa se adaptan a la evolución del entorno exterior.

- **Estandarizado:** Normativa clara y aceptada por consenso sobre los elementos que debe ofrecer el lenguaje. La estandarización facilita que las aplicaciones se puedan portar de unos compiladores a otros sin cambios. El lenguaje, desde su primera estandarización en el año 83 ha experimentado otras en el 95 y 2005 que han añadido nuevas funcionalidades en consonancia con los tiempos. Por ejemplo, se introdujo la programación orientada a objetos o mejoras en la programación en tiempo real.

# 2.3. Protocolos para la comunicación de redes: Ethernet, IP, UDP y DHCP

Como ya se ha comentado, una de las características fundamentales que han hecho decantarse por el tipo de cámara digital escogida es su interfaz de conexión Ethernet. A grandes rasgos, Ethernet es un estándar para la comunicación de dispositivos en redes de área local que define tanto las características de cableado como el formato de los datos.

El estándar para transmisión de vídeo y control de dispositivos GigE Vision que se ha implementado en este proyecto cuenta con Ethernet como la base sobre la cual se construyen el resto de las comunicaciones. Para llegar al nivel de transmisión de los datos que el protocolo GigE Vision pretende manejar, existen una serie de **capas intermedias** que son implementadas por protocolos que se encuentran fuera de la especificación de GigE Vision y que son de uso extremadamente común en redes de comunicaciones: IP y UDP. Todos estos protocolos se *apilan, superponen o encapsulan* unos dentro de otros formando una **pila** que responde a lo que denominamos **arquitectura de protocolos**, cuyos principios básicos se exponen a continuación en el Apartado 2.3.1, "Arquitectura de protocolos". Más adelante se describirán los protocolos Ethernet (Apartado 2.3.2, "Ethernet"), IP (Apartado 2.3.3, "IP") y UDP (Apartado 2.3.4, "UDP").

Por otro lado, se cuenta con el protocolo DHCP, cuya principal funcionalidad es la de otorgar a los clientes de una red basada en el protocolo IP sus parámetros de configuración de forma automática. Se describirá también este protocolo (Apartado 2.3.5, "DHCP") pues ha sido incluido en este proyecto.

## 2.3.1. Arquitectura de protocolos

La arquitectura de protocolos es una técnica para estructurar jerárquicamente la funcionalidad de un sistema de comunicaciones en capas de elementos hardware y software utilizando protocolos estructurados que facilita el intercambio de datos entre sistemas (Stallings 2004, cap. 2).

En el intercambio de datos entre computadores, terminales y/u otros dispositivos de procesamiento, los procedimientos involucrados pueden llegar a ser bastante complejos. Pongamos como ejemplo el sistema montado en este proyecto y en el cual el objetivo final es la obtención de una imagen entre el computador y la cámara. Sin adelantar demasiados detalles, el computador debe tener la iniciativa en la petición de la imagen, por lo que iniciará el diálogo. Está claro que debe haber un camino directo o a través de una red de comunicación que los conecte y que se deben llevar a cabo varias tareas:

- El computador debe de activar un camino directo de datos o proporcionar a la red de comunicación la identificación de la cámara.
- El computador debe cerciorarse de que la cámara está preparada para recibir la petición de captura.
- El computador debe estar preparado para recibir la imagen.
- El formato de la imagen puede no concordar con el deseado en el computador y debe de haber una aplicación que la manipule.

Esta aproximación simplista de la realidad tiene como objetivo plasmar que debe de haber un alto grado de cooperación entre el computador y la cámara. En lugar de implementar toda la lógica para llevar a cabo la comunicación en un único módulo, **el problema se divide en subtareas (capas)**, cada una de las cuales se realiza por separado. Cada capa de la pila de capas realiza el subconjunto de tareas relacionadas entre sí que son necesarias para comunicar con el otro sistema. Por lo general, **las funciones mas básicas se dejan a la capa inmediatamente inferior**, olvidándose la capa superior de los detalles de esas funciones. Además, **cada capa proporciona un conjunto de servicios a la capa inmediatamente superior**.

Para que haya comunicación, se necesitan **dos entidades en las cuales deben existir el mismo conjunto de funciones en cada capa**. La comunicación se consigue haciendo que las entidades correspondientes (pares) intercambien información. Estas capas pares se comunican enviándose bloques de datos que verifican una serie de reglas o convenciones denominadas **protocolo**. Aspectos claves de un protocolo son:

- **Sintaxis:** Formato de los datos.
- **Semántica:** Significado de esos datos.
- **Temporización:** Velocidades de transmisión y secuenciación de los datos transmitidos.

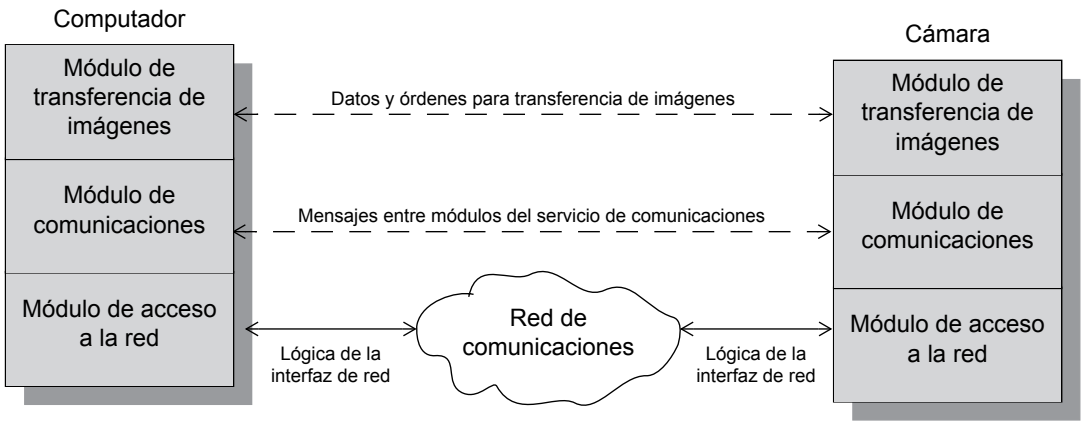


**Figura 2.2: Arquitectura de protocolos simplificada para la captura de imágenes.**

En la cúspide de esta pila se encuentran los datos que maneja la aplicación.

Se propone un ejemplo: un caso cercano al ámbito del proyecto en que los datos son una imagen. Cada mensaje transmitido tiene una limitación en el número de datos que pueden transportar. Sin embargo, en muchas ocasiones la imagen desborda esta capacidad y deberá ser dividida en trozos, cada uno de los cuales es transmitido desde la cámara al computador.

Para poder llevar a cabo su función, cada protocolo añade unos datos adicionales especificados en su sintaxis. Estos datos añadidos por cada capa son lo que se denominan **cabeceras** (Figura 2.3).

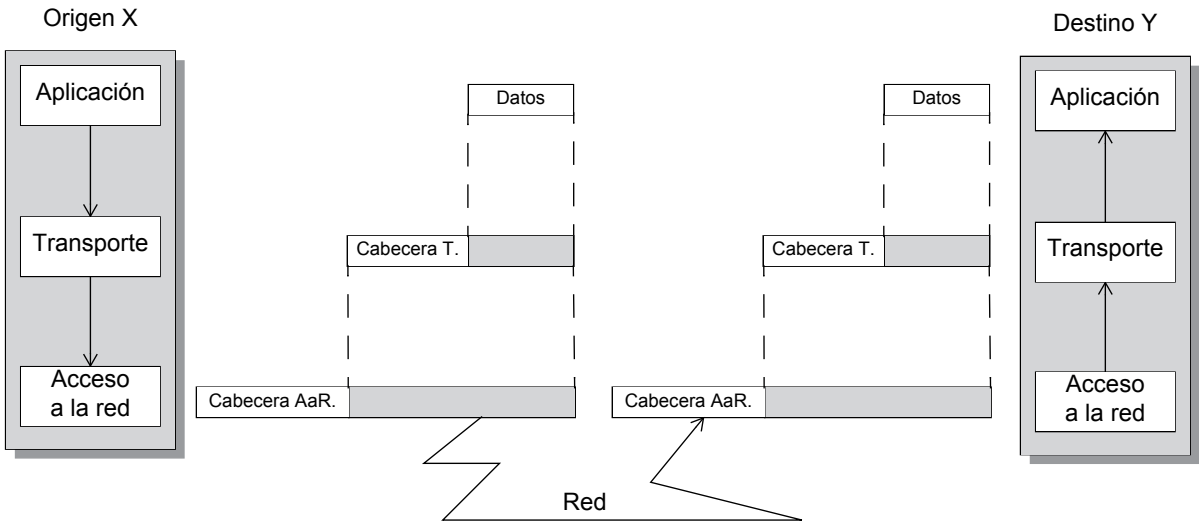


**Figura 2.3: Funcionamiento de una arquitectura de protocolos.**

Las cabeceras añadidas junto con los datos de la imagen son transferidos por la red desde la capa inferior.

Existen distintas arquitecturas que permiten la comunicación en redes. Una de ellas y que se emplea como **referencia** es la denominada **modelo OSI**. OSI es una especificación teórica que sirve como marco de referencia para la definición de arquitecturas de interconexión. **Propone una pila de 7 capas y define la funcionalidad de cada una de ellas**. Sin embargo, se trata de un modelo que fue propuesto demasiado tarde, cuando ya se habían implementado las tecnologías que debían haberlo tomado como base, por lo que su utilidad es más la de ser una referencia que algo a implementar. De hecho, es en realidad la familia de protocolos de Internet la que se considera el estándar común y sobre la que se asienta el protocolo GigE Vision.

En la Figura 2.4 se muestran los protocolos de comunicaciones empleados en el proyecto junto con la capa UDP/IP a la que corresponden y estas últimas a su vez relacionadas con las capas de referencia OSI a que corresponden. En cada uno de ellos se explica la función que tienen.

GVCP (*GigE Vision Control Protocol*) y GVSP (*GigE Vision Stream Protocol*) son los protocolos propios definidos en el estándar GigE Vision. En los siguientes apartados se explicarán los protocolos Ethernet, IP, UDP y DHCP. Tanto DHCP como GVCP y GVSP se apilan sobre UDP.

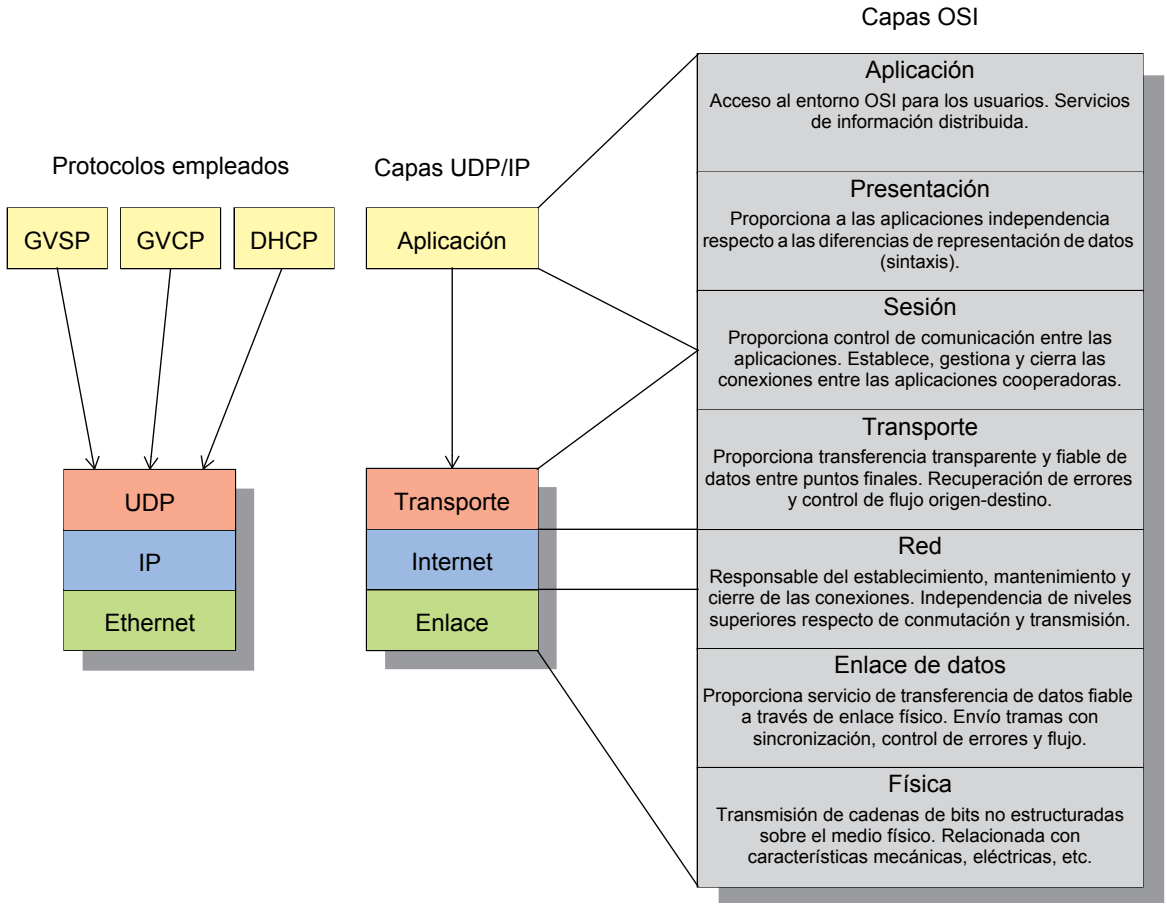


**Figura 2.4: Protocolos de comunicación empleados en el proyecto referidos a la pila de protocolos UDP/IP y a su vez al modelo de referencia OSI.**

## 2.3.2. Ethernet

Ethernet es una familia de tecnologías para redes de ordenadores de área local (LANs) desarrollada por el comité de estándares 802.3 (Stallings 2004, cap. 16). Es la más empleada dentro del mundo de las LANs. En referencia a las capas OSI, Ethernet se ocupa de las capas física y de enlace de datos.

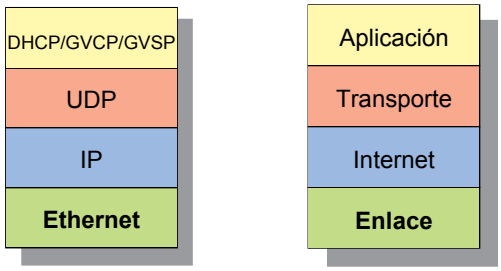


**Figura 2.5: Ubicación de Ethernet en la pila de protocolos UDP/IP y en la pila OSI.**

Capa física

Define los siguientes aspectos de la red (Vallejo 2008):

- **Medio de transmisión y topología:** Existen varias alternativas. Una de ellas es la *fibra óptica*. En el caso de este proyecto, tanto la cámara como el PC disponen como interfaz física del conector denominado RJ-45. Diferentes tipos de cableado se puede emplear

con este conector. Estos se definen en estándares como el definido en TIA/EIA-568 que define las categorías de cableado *UTP/STP*.

**Figura 2.6: Conector RJ-45.**

- **Codificación/decodificación de señales:** Si se emplea el cableado UTP/STP con RJ-45 o fibra óptica, la codificación que se realiza de los datos es distinta. Este es solo un ejemplo de los objetivos de la codificación y decodificación de señales.
- **Generación/eliminación de preámbulos:** A efectos de sincronización de relojes en emisor y receptor.
- **Transmisión/recepción de bits.**

Capa de enlace de datos

Define los siguientes aspectos de la red (Vallejo 2008):

- **Control de acceso al medio:** Al enviar datos se debe tener en cuenta que puede haber varios emisores que difundan sus mensajes y que si esto sucede simultáneamente, el mensaje no llega. Ethernet emplea **CSMA/CD** (*Carrier Sense Multiple Access with Collision Detection*), que a grandes rasgos implica que se utiliza un medio de acceso múltiple y que la estación que desea transmitir previamente escucha el canal antes de hacerlo.
- **Delimitación de tramas:** Mecanismos para diferenciar en el medio unas tramas de otras.
- **Cálculo de códigos de control y descarte de tramas erróneas:** Debido a la naturaleza del medio de transmisión, pueden producirse errores que modifiquen los bits transmitidos.
- **Direccionamiento:** Especificación del origen y destino de la trama.

Derivado de estas características, la trama Ethernet necesaria es la que aparece en la Figura 2.7.

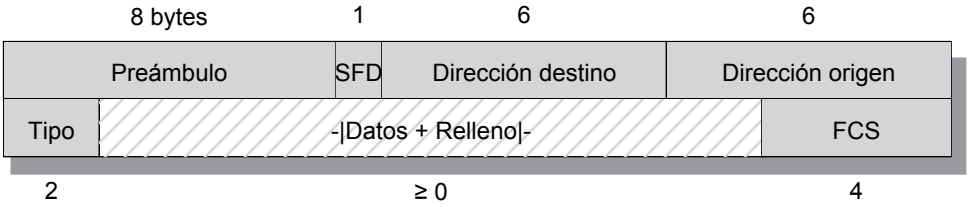


**Figura 2.7: Formato de la trama Ethernet.**

Como Ethernet es (dentro de la arquitectura de protocolos) el nivel más cercano al hardware, es posible que algunos de los campos no tengan que ser especificados por el software sino que sean manejados por el propio hardware.

Cada campo tiene el siguiente significado (Stallings 2004, cap. 16):

- **Preámbulo** (*preamble*) [7 bytes]: el receptor usa 7 octetos de bits ceros y unos alternados para establecer la sincronización entre emisor y receptor.
- **Delimitador de comienzo de trama** (*SFD - Start of Frame Delimiter*) [1 byte]: Se introduce la secuencia 10101011, que indica el comienzo real de la trama y posibilita que el receptor pueda localizar el primer bit del resto de la trama.
- **Dirección de destino** (*destination address*) [6 bytes]: Especifica la estación o estaciones a que va dirigida la trama. Puede tratarse de una única dirección física, una dirección de grupo o una dirección global. Se especifica mediante la dirección MAC, un identificador de 48 bits que define de forma única a una tarjeta o dispositivo de red.
- **Dirección de origen** (*source address*) [6 bytes]: Especifica la estación que envió la trama mediante una dirección MAC.
- **Tipo** (*type*) [2 bytes]: Indica el protocolo que se lleva encapsulado en el campo de datos. Por ejemplo, para IP es 0x0800.
- **Datos** [≥ 0 bytes]: Los datos que transporta la trama. Longitud múltiplo de 8 bits.
- **Relleno** [≥ 0 bytes]: Octetos añadidos para asegurar que la trama sea lo suficientemente larga como para que la técnica de Detección de Colisiones (CD) funcione correctamente.
- **Secuencia de comprobación de trama** (*FCS - Frame Check Sequence*) [4 bytes]: Comprobación de Redundancia Cíclica de 32 bits, calculada teniendo en cuenta todos los campos excepto el preámbulo, el SFD y el FCS.

Especificaciones del estándar Ethernet

La especificación original de Ethernet definía una velocidad de transmisión de datos de 10 Mbps. Modificaciones en aspectos como el cableado o la forma en que se envían los datos han conducido a la aparición de *Fast Ethernet* a 100 Mbps y *Gibabit Ethernet* a 1000 Mbps.

Aunque el estándar GigE Vision está pensado para ser usado con Gigabit Ethernet, puede ser implementado para cualquier velocidad de Ethernet. En este proyecto, la tarjeta de red del PC es **Fast Ethernet**, así que el sistema se adaptará a esta velocidad.

### 2.3.3. IP

IP (*Internet Protocol*) es el protocolo de interconexión de redes más utilizado (Stallings 2004, cap. 18). La versión que se va a examinar es la IPv4, si bien se espera que en años próximos le suceda la IPv6. IP ocupa la capa de red respecto a la referencia OSI.

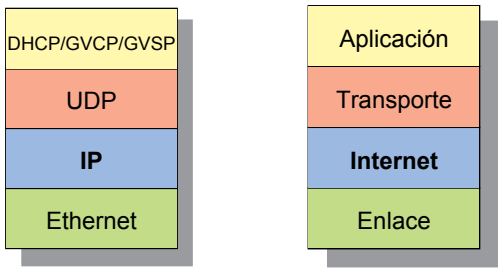


**Figura 2.8: Ubicación de IP en la pila de protocolos UDP/IP y en la pila OSI.**

Los datos en una red basada en IP son enviados en bloques conocidos como *paquetes* o *datagramas*.

IP ofrece un **servicio**:

- **No orientado a la conexión:** El dispositivo en un extremo de la comunicación transmite los datos al otro sin asegurarse de que el receptor esté disponible.
- **No confirmado:** Los datos pueden perderse, duplicarse o desordenarse y la red no detecta estos hechos.

Deben de ser las capas superiores las que eviten estas limitaciones si así lo desearan. La única comprobación de seguridad que realiza el protocolo IP es sobre sus cabeceras mediante sumas de comprobación y no sobre los datos transmitidos.

Las cabeceras IP contienen las direcciones de las máquinas de origen y destino (direcciones IP). Estas direcciones se emplean por los enrutadores para saber por qué tramo de red deben enviar los paquetes.

El protocolo se define mejor mediante la explicación de los campos presentes en la Figura 2.9.

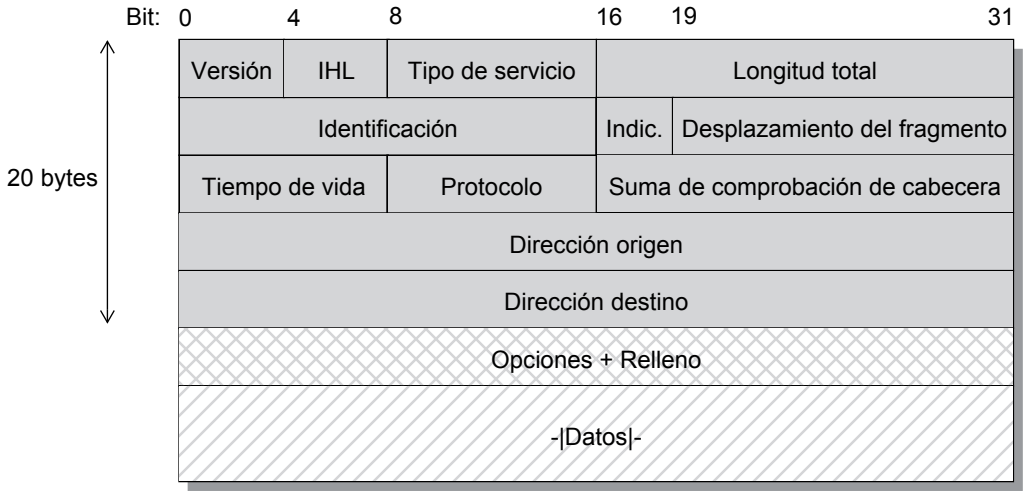


**Figura 2.9: Formato de la trama IPv4.**

- **Versión** (*version*) [Bits 0-3 (4)]: Indica el número de la versión del protocolo, para permitir la evolución del mismo. En este proyecto se emplea IPv4, por lo que su valor es 4.

- **Longitud de cabecera** (*IHL - Internet Header Length*) [Bits 4-7 (4)]: Longitud de cabecera expresada en múltiplos de 32 bits. El valor mínimo es de 5, correspondiente a una longitud de cabecera de 20 bytes.
- **Tipo de servicio** (*type of service*) [Bits 8-15 (1 byte)]: Especifica los parámetros de prioridad (bits 8-10), retardo (bit 11), rendimiento (bit 12) y fiabilidad (bit 13). Este campo se utiliza muy raramente.
- **Longitud total** (*total length*) [Bits 16-31 (2 bytes)]: Longitud total del datagrama en bytes (cabecera + datos).
- **Identificador** (*identification*) [Bits 32-47 (2 bytes)]: Número de secuencia que, junto a la dirección de origen y destino y el protocolo usuario se utiliza para identificar de forma única un datagrama. Si el datagrama se fragmenta en la red, cada uno de los componentes llevará la misma identificación.
- **Indicadores** (*flags*) [Bits 48-50 (3 bits)]: El bit 49 prohibe la fragmentación del datagrama. Si al enviarlo por la red fuera necesario fragmentarlo pero se ha especificado que no se puede, el paquete se pierde. El bit 50 indica que no es el último datagrama de una serie de fragmentados.
- **Desplazamiento del fragmento** (*fragment offset*) [Bits 51-63 (13 bits)]: Indica el lugar en el cual se insertará el fragmento actual dentro del datagrama completo medido en unidades de 64 bits. Por ello, la cantidad de datos de cada fragmento (salvo el último, que puede ser de menos) debe ser múltiplo de 64 bits.
- **Tiempo de vida** (*TTL - Time To Live*) [Bits 64-71 (1 byte)]: Número máximo de saltos que puede dar un datagrama en la red. Cada dispositivo de encaminamiento que procesa el datagrama debe decrementar este campo en una unidad.
- **Protocolo** (*protocol*) [Bits 72-79 (1 byte)]: Identifica el protocolo de la capa de red inmediatamente superior que va a recibir el campo de datos en el destino. Así, este campo sirve para identificar el siguiente tipo de cabecera presente en el paquete después de la cabecera IP. En el proyecto sólo se apilará el protocolo UDP sobre IP y este campo valdrá siempre 17.
- **Suma de comprobación de cabecera** (*header checksum*) [Bits 80-95 (2 bytes)]: Código de detección de errores aplicado solamente a la cabecera. Ya que algunos campos de la cabecera pueden cambiar durante el viaje del paquete por la red (por ejemplo el tiempo de vida), este valor se verifica y recalcula en cada dispositivo de encaminamiento. Para calcularlo, se realiza la suma complemento a uno de todas las palabras de 16 bits de la cabecera.
- **Dirección de origen** (*source address*) [Bits 96-127 (4 bytes)]: Dirección IP de origen.
- **Dirección de destino** (*destination address*) [Bits 128-159 (4 bytes)]: Dirección IP de destino.
- **Opciones** (*options*) [variable]: Contiene las opciones solicitadas por el usuario que envía los datos. Es un campo no obligatorio que no ha sido empleado en este proyecto.
- **Relleno** (*padding*) [variable]: Si las opciones IP (en caso de existir) no ocupan un múltiplo de 32 bits, se rellena con ceros.
- **Datos** [variable]: Datos que encapsula el protocolo. La longitud debe ser múltiplo de 8 bits y el máximo teórico de datos más cabecera es de 65535 bytes. No obstante, la longitud máxima real viene dada por la máxima cantidad de datos que nos permita la capa inmediatamente inferior, en este proyecto la capa Ethernet.

### 2.3.4. UDP

UDP (*User Datagram Protocol*) es un protocolo de la capa de transporte (modelo de referencia OSI) que proporciona un servicio no orientado a la conexión para los procedimientos de la capa de aplicación. Por tanto, UDP es básicamente un servicio no fiable: no se garantizan la entrega y la protección contra duplicados (Stallings 2004, cap. 20).

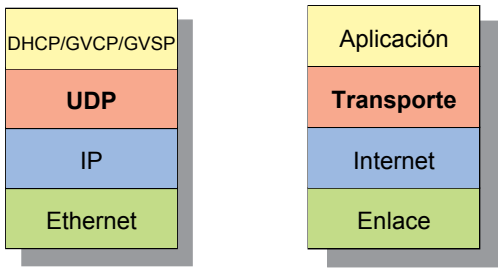


**Figura 2.10: Ubicación de UDP en la pila de protocolos UDP/IP y en la pila OSI.**

La alternativa por excelencia a UDP es TCP. TCP es un protocolo que permite crear conexiones entre computadoras a través de las cuales se envía un flujo de datos. El protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron.

Como es previsible, TCP no son todo ventajas. Este modo de funcionamiento introduce una sobrecarga que UDP no genera y que hace que UDP sea aconsejable en aplicaciones en las que **el flujo de datos no admite demoras**.

Este proyecto, con un sistema de transmisión de video es uno de esos en los que TCP debe desecharse en favor de UDP. No obstante, esto obliga a valorar si las posibles pérdidas que se pudieran producir son asumibles. En caso de que no lo fueran, se pueden utilizar otros recursos como la **retransmisión**.

UDP se sitúa sobre IP. Ya que es no orientado a la conexión, UDP tiene muy pocas tareas que llevar a cabo. Esencialmente, incorpora a IP la **capacidad de un direccionamiento de puerto** que permita distinguir entre las distintas aplicaciones que pueden estar conectadas al mismo dispositivo.

La cabecera y los campos UDP aparecen en la Figura 2.11.

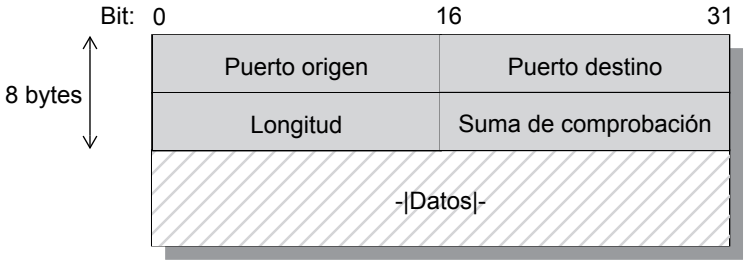


**Figura 2.11: Formato de la trama UDP.**

- **Puerto origen** (*source port*) [Bits 0-15 (2 bytes)]: Identifica el proceso de origen.
- **Puerto destino** (*destination port*) [Bits 16-31 (2 bytes)]: Identifica el proceso de destino.
- **Longitud** (*length*) [Bits 32-47 (2 bytes)]: Número de bytes en cabecera y datos, con un máximo teórico de 65535 bytes. En realidad, el límite viene impuesto por la cantidad de datos que permita la capa inmediatamente inferior (IP).
- **Suma de comprobación** (*checksum*) [Bits 48-63 (2 bytes)]: Mismo algoritmo al empleado con IP. La suma de comprobación se aplica al segmento UDP entero más una pseudocabecera formada por campos de IP. Este checksum es opcional.
- **Datos** [variable]: Datos que encapsula el protocolo con longitud múltiplo de 8 bits.

### 2.3.5. DHCP

DHCP (*Dynamic Host Configuration Protocol*) es un protocolo de configuración de red para equipos anfitriones que se encuentren en una red IP (RFC2131 1997).

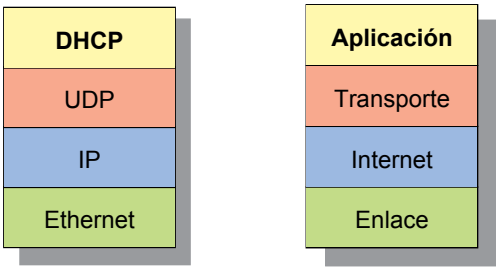


**Figura 2.12: Ubicación de DHCP en la pila de protocolos UDP/IP y en la pila OSI.**

Los dispositivos conectados a redes IP necesitan ser configurados antes de poder comunicar con otros equipos. La mínima configuración necesaria para comunicarse consiste en:

- Dirección IP.
- Puerta de enlace predeterminada.
- Máscara de subred.

DHCP emplea el modelo *cliente-servidor* permitiendo **obtener para el cliente estos parámetros** de configuración y muchos otros dados por el servidor sin necesidad de realizarlo de forma manual. Además permite mantener un **control sobre qué dispositivos se encuentran conectados a la red** y **que recursos tienen asociados**.

El servidor cuenta con una serie de parámetros/recursos que puede asignar. Dependiendo de la forma en que asocie las direcciones IP, se tienen tres modelos:

- **Asignación manual o estática:** El servidor asigna una dirección IP permanente otorgada por el administrador en función de la dirección MAC del cliente. Cuenta para ello con una tabla de correspondencias.
- **Asignación dinámica:** El servidor asigna una dirección IP dentro de un rango que tiene establecido y por periodo limitado.
- **Asignación automática:** El servidor asigna una dirección IP permanente.

La configuración de red en DHCP se consigue mediante el **establecimiento de un diálogo** de mensajes entre el cliente y servidor. El formato de los mensajes DHCP está basado en el formato de los mensajes del protocolo BOOTP, del cual se considera una evolución.

La Figura 2.13 muestra el formato de mensaje DHCP.

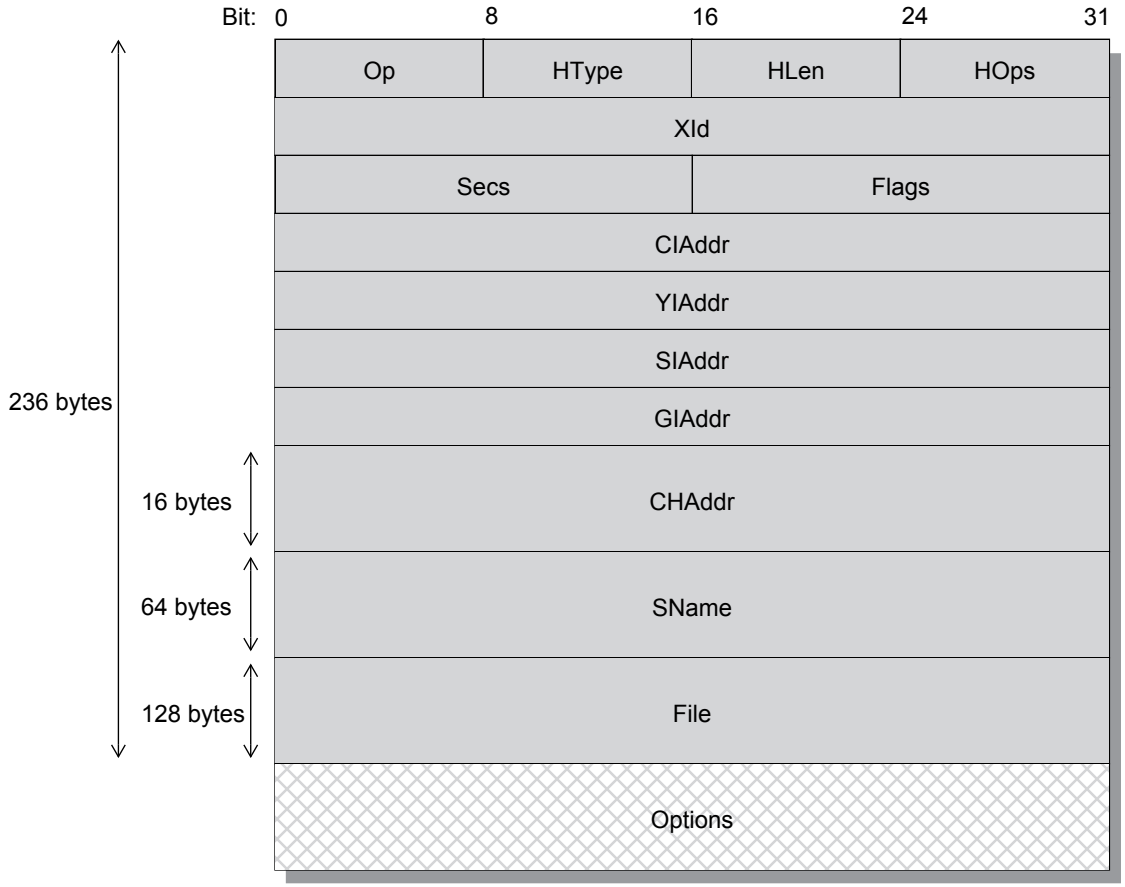


**Figura 2.13: Formato de mensaje DHCP.**

- **Op** [Bits 0-7 (1 byte)]: Indica el tipo de mensaje: 1 BOOTREQUEST, 2 BOOTREPLY.
- **HType** [Bits 8-15 (1 byte)]: Tipo de hardware: Ethernet (1), Frame Relay (15), Infiniband (32), etc.
- **HLen** [Bits 16-23 (1 byte)]: Longitud de la dirección hardware. 6 para la dirección MAC (6 bytes).
- **HOps** [Bits 24-31 (1 byte)]: Utilizado por *relay agents*. No interesa en el proyecto (dejarlo a 0).
- **XId** [Bits 32-63 (4 bytes)]: Número aleatorio escogido por el cliente y usado por el cliente y servidor para relacionar los mensajes y respuestas que se envían entre ellos.
- **Secs** [Bits 64-79 (2 bytes)]: Tiempo en segundos desde que el cliente pidió/renovó la dirección.
- **Flags** [Bits 80-95 (2 bytes)]: Bit 80 puesto a 1 significa mensaje de difusión. El resto debe estar puesto a 0.
- **CIAddr** [Bits 96-127 (4 bytes)]: IP del cliente. Sólo se fija si el cliente ya cuenta con una IP previamente. Por ejemplo, esta es la situación que se da cuando se renueva una dirección.
- **YIAddr** [Bits 128-159 (4 bytes)]: Dirección IP que el servidor ofrece al cliente.
- **SIAddr** [Bits 160-191 (4 bytes)]: IP del servidor.
- **GIAddr** [Bits 192-223 (4 bytes)]: Para uso con relay agent.

- **CHAddr** [Bits 224-351 (16 bytes)]: Dirección MAC del cliente.
- **SName** [Bits 352-863 (64 bytes)]: Nombre del servidor.
- **File** [Bits 864-1887 (128 bytes)]: Nombre del archivo de boot. Empleado para arrancar máquinas.
- **Options** [variable]: Las opciones DHCP permiten configurar muchos otros parámetros. Transportan información entre cliente y servidor que puede ser útil para la configuración. Las opciones DHCP siguen el mismo esquema que las llamadas *vendor extensions* de BOOTP. Cuentan con un primer campo denominado **Magic Cookie** [Bits 1888-1919 (4 bytes)] que identifica el modo en que los siguientes datos deben ser interpretados. Su valor es 0x63825363. A continuación se encuentran las opciones en sí. Su número es variable y siguen el siguiente esquema de la Figura 2.14.

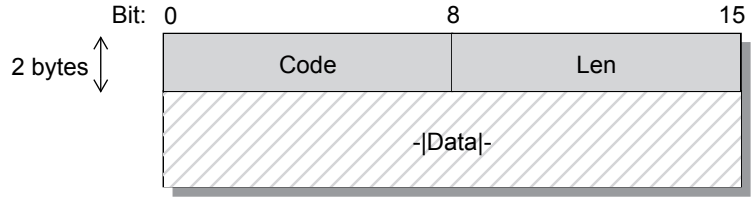


**Figura 2.14: Formato de la opción DHCP.**

- **Code** [Bits 0-7 (1 byte)]: Identifica a la opción.
- **Len** [Bits 8-15 (1 byte)]: Número de bytes de la sección de datos.
- **Data** [variable]: Los datos enviados. Tienen la longitud indicada en Len y son interpretados de acuerdo a Code.

La cantidad de opciones definidas está cerca de los dos centenares. Se definen dependiendo de las peculiaridades de cada protocolo que las utilice. Las hay desde para protocolos de correo hasta servidores TFTP de transferencia de archivos. Sin embargo, las cámaras GigE Vision no hacen uso de opciones muy fuera de lo común. Las opciones empleadas son las siguientes (RFC1533 1993; RFC Source Book 2011):

- **Subnet mask** (Código: 1 // Longitud: 4): Máscara de subred del cliente.
- **Router** (Código: 3 // Longitud: 4 o más): Una o más direcciones IP con los routers a utilizar por el cliente. Se listan por orden de preferencia.
- **Host name** (Código: 12 // Longitud: 1 o más): Especifica el nombre del cliente.
- **IP address lease time** (Código: 51 // Longitud: 4): Tiempo por el cual se concede la dirección IP.
- **DHCP message type** (Código: 53 // Longitud: 1): Tipo de mensaje DHCP (se verá más adelante).
- **Server identifier** (Código: 54 // Longitud: 4): IP del servidor. Permite al cliente distinguir entre las ofertas de los servidores.
- **Parameter request list** (Código: 55 // Longitud: 1 o más): Opción empleada por un cliente DHCP para pedir al servidor los valores de los parámetros de configuración especificados. La lista de parámetros pedidos se especifica mediante una ristra de bytes en la que cada byte es una opción DHCP.
- **Renew time value** (Código: 58 // Longitud: 4): Intervalo de tiempo desde la asignación de la dirección IP hasta que el cliente pasa al estado de *renewing* en que debe ocuparse de renovar la dirección.

- **Rebinding time value** (Código: 59 // Longitud: 4): Intervalo de tiempo desde la asignación de la dirección IP hasta que pasa al estado *rebinding*.

A continuación se describirá el diálogo establecido entre el cliente y el servidor para la configuración de red. DHCP cuenta con cuatro tipos de mensajes distintos, cuya secuencia es la que sigue:

1. **DHCP Discovery:** Mensaje enviado por el cliente DHCP al servidor (se envía en *modo difusión/broadcast*) para que este le asigne una dirección IP y otros parámetros de red.
2. **DHCP Offer:** Cuando al servidor le llega un mensaje de DHCP Discovery, responde al cliente (modo *unicast*) con una dirección IP que le reserva, asociándola a su dirección MAC. Además, le envía todos los parámetros que el cliente hubiese solicitado.
3. **DHCP Request:** El cliente puede haber recibido varias ofertas DHCP Offer. Selecciona una de ellas y envía en modo difusión un DHCP Request. De esta forma, aquellos servidores que han sido rechazados pueden recuperar la dirección IP que habían reservado.
4. **DHCP Acknowledge:** Cuando el servidor seleccionado recibe el mensaje DHCP Request, responde (modo unicast) con un DHCP Ack. Este mensaje vuelve a incluir los parámetros enviados en el DHCP Offer.

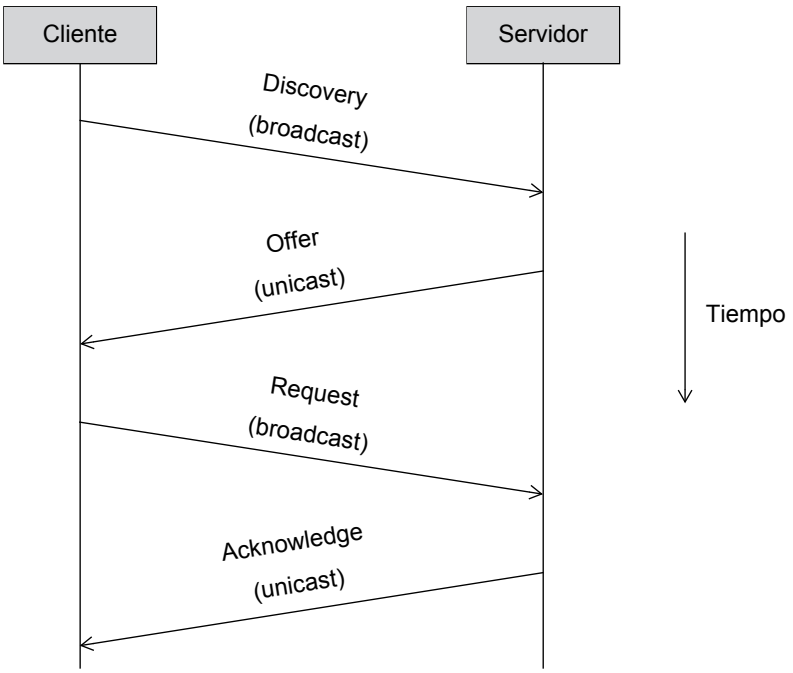


**Figura 2.15: Intercambio de mensajes entre cliente y servidor en el diálogo DHCP.**

# 3. Estándar GigE Vision

## 3.1. Introducción y propósito

GigE Vision es una interfaz de comunicación para aplicaciones de visión introducida en 2006 para su empleo en el ámbito de las cámaras de alto rendimiento industriales. Está basada en la tecnología Gigabit Ethernet (Apartado 2.3.2, "Ethernet"). Proporciona un método de conexión sencillo y de gran velocidad entre un dispositivo GigE Vision y una tarjeta de red empleando medios físicos de conexión Ethernet que son de bajo coste y permiten la transmisión de datos a largas distancias.

El estándar pretende unificar los protocolos que se estaban empleando en las cámaras industriales. Con GigE Vision, hardware y software de distintos fabricantes pueden interoperar sin problema.

El estándar fue iniciado por un grupo de 12 compañías, número que se ha visto aumentado en gran medida en los últimos años por la inclusión de nuevos miembros. Entre estas compañías se encuentran algunas de las más importantes dentro del mundo de la visión por computador, como son *Allied Vision*, *Dalsa Corp.*, *Kappa opto-electronics*, *National Instruments* o *Toshiba TELI Corp.* (Machine Vision Online 2011). La *Automated Imaging Association (AIA)* supervisa el desarrollo y administración del estándar.

El estándar es considerado abierto por algunas organizaciones como la ITU. Sin embargo, sólo se puede disponer de él tras firmar un acuerdo de no divulgación. Por tanto, no es posible escribir software *open source* pues podría revelar los detalles del estándar. Esto conlleva además limitaciones a la hora de plasmar en esta memoria del proyecto detalles de implementación. Por ello el objetivo de este capítulo es dar una visión general del funcionamiento del estándar sin entrar en detalles concretos de cómo se hacen realmente las cosas (campos de tramas, nombres y direcciones concretos de registros, etc.).

Algunas de las características técnicas que ofrece GigE Vision se muestran en la Tabla 3.1.

**Tabla 3.1: Características técnicas de la interfaz GigE Vision.**

| | |
|---|---|
| **Velocidad** | Ancho de banda de 1000 Mbps |
| **Longitud cableado** | Velocidad máxima garantizada hasta los 100 m |
| **Estandarizado** | Cables de conexión de bajo coste CAT5e y CAT6 |
| **Penetración** | Emplea Ethernet, tecnología en constante crecimiento |
| **Coste** | Hardware y cables estándar permiten una integración de bajo coste |

En el estándar se definen las características que los sistemas que soporten GigE Vision deben tener en cuenta. Se especifica si son de obligado cumplimiento o son opcionales. Sin embargo, el estándar no llega hasta los últimos términos en la definición del funcionamiento de las aplicaciones y dispositivos. Proporciona el marco en el que los fabricantes y desarrolladores deben moverse, pero no lo especifica hasta el último detalle. Por ejemplo, el fabricante de una cámara GigE Vision puede definir sus propios códigos de errores o la estructura de la memoria.

A pesar de que el estándar se refiere específicamente a Gigabit Ethernet, puede ser utilizado en cualquier red Ethernet de cualquier velocidad. De hecho, en este proyecto se ha trabajado a 100 Mbps debido a las limitaciones de la tarjeta de red del PC controlador.

### 3.1.1. Sistemas GigE Vision

El estándar está definido para dar soporte a distintas redes de dispositivos (GigE Vision Standard 2010): desde la más simple en que un dispositivo de video se conecta al computador que acoge la aplicación mediante el uso de un cable cruzado, a otras que cuentan con varias cámaras, unidades de proceso y componentes de interconexión Ethernet como routers.

Así mismo, habla de forma genérica sobre dispositivos, dando a entender que no son sólo las cámaras el objeto del estándar y que se pueden conectar otros. Algunos de estos podrían ser servidores de vídeo, conversores de formato a HDMI, etc. para los cuales este protocolo de transporte sería útil. No obstante, en este proyecto no se hará distinción alguna entre dispositivo y cámara, pues la cámara es el destinatario más común para el estándar y en el caso que nos ocupa será lo único que conectaremos.

La Figura 3.1 muestra el esquema básico en que contamos con una cámara y un computador que ejecuta la aplicación de control y recibe los datos. Este será el caso de nuestro sistema, con la peculiaridad de que el controlador será un PC que ejecuta el sistema operativo MaRTE OS.

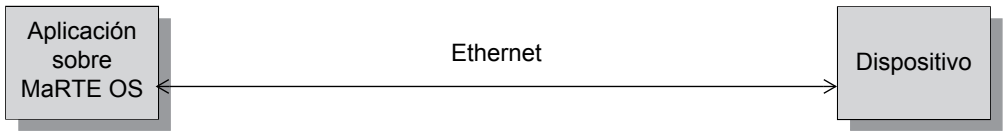


**Figura 3.1: Sistema común para control y recepción de imágenes.**

También se pueden conectar estos dos componentes introduciendo redes Ethernet adicionales de por medio (Figura 3.2). No es problema del estándar sino de los protocolos de red y enlace el que los paquetes lleguen correctamente de un extremo al otro. Sin embargo, los paquetes si que podrían llegar desordenados y la latencia de transmisión verse aumentada.


**Figura 3.2: Sistema común para control y recepción de imágenes con red de por medio.**

Existen ejemplos de sistemas más complejos que se comunican empleando el protocolo GigE Vision. Como se dijo anteriormente, no sólo una cámara puede ser conectada a la red. La Figura 3.3 nos muestra un ejemplo de ello.

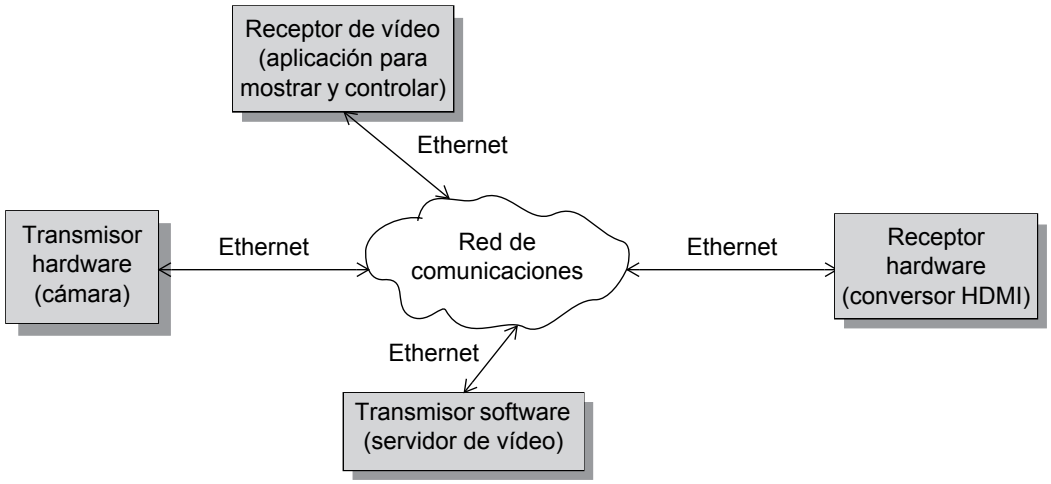


**Figura 3.3: Sistema avanzado de vídeo sobre Ethernet.**

## 3.1.2. Estructuración y funcionamiento

El estándar GigE Vision se compone de tres partes que corresponden con las funciones principales que desarrolla:

- **Descubrimiento de dispositivos:** Involucra la fase en que la aplicación debe descubrir y enumerar los dispositivos GigE Vision conectados a la red.
- **Control del dispositivo (GigE Vision Control Protocol, GVCP):** Describe el protocolo que permite el intercambio de mensajes de control con los que modificar el comportamiento de la cámara y sus parámetros.
- **Obtención de imágenes (GigE Vision Stream Protocol, GVSP):** Describe el protocolo que transporta las imágenes y su información asociada a través de la red

Para lograr sus objetivos, el estándar crea dos protocolos: GVCP y GVSP que se apilan sobre el UDP. Durante la fase de descubrimiento de dispositivos, se emplea el protocolo DHCP combinado con GVCP para dotar de dirección IP a la cámara.

En las secciones siguientes se describirá de forma más detallada la fase de descubrimiento de dispositivos y los protocolos GVCP y GVSP. Se tratará sobre su funcionalidad e implementación dentro del estándar.

Por otro lado y a modo de reseña, comentar que los dispositivos GigE Vision deben dar soporte al protocolo de nivel de transporte ICMP, más conocido por su utilidad en la herramienta *ping*. Ping permite comprobar la conectividad entre el host desde el que se envía y la cámara mediante el paquetes ICMP.

# 3.2. Descubrimiento de dispositivos

La fase de descubrimiento de dispositivos (en el estándar *Device Discovery*) cubre la secuencia de eventos requerida para que un dispositivo obtenga una dirección IP válida y de a conocer sus características básicas a la aplicación de control.

Una visión esquemática del sistema se muestra en la Figura 3.4. En un primer momento, la cámara y la aplicación no tienen conocimiento mutuo.

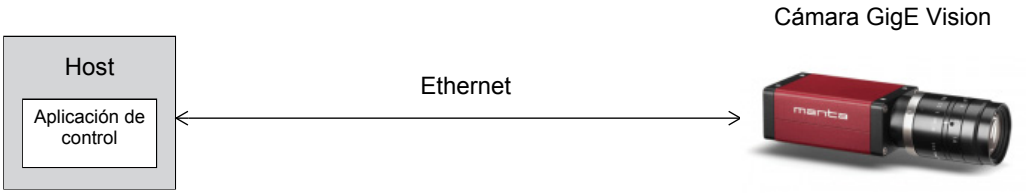


**Figura 3.4: Visión esquemática del sistema que se maneja en el proyecto.**

## 3.2.1. Fases del descubrimiento

La fase de descubrimiento cubre en primer lugar la configuración IP y a continuación se ejecuta la fase de enumeración de dispositivos. Se describe detalladamente cada una de ellas.

### 3.2.1.1. Configuración IP

Iniciada por la cámara, busca obtener los parámetros de direccionamiento que la permitan comunicarse dentro de una red IP:

- Dirección IP.
- Máscara de subred.
- Puerta de enlace predeterminada.

La conexión entre la cámara y la aplicación solo puede realizarse cuando ambas tienen estos parámetros configurados correctamente. Tres son las alternativas mediante las cuales se pueden conseguir estos parámetros. Cuando el dispositivo es enchufado, lo primero que hace es buscar su configuración IP ejecutando en orden estas alternativas. En caso de que alguna de ellas falle, se pasa a la siguiente. Si todas fallan, no es posible obtener la configuración IP.

1. **IP persistente** (*Persistent IP*): La implementación de esta configuración es opcional. En la memoria de la cámara deben encontrarse almacenados los datos de direccionamiento. Antes de proceder a usarlos, el dispositivo debe cerciorarse mediante el protocolo ARP de que no existen conflictos entre otros dispositivos conectados a la red que ya pudieran hacer uso de la misma IP. En ese caso, el dispositivo debería señalizarlo mediante, por ejemplo, el parpadeo del LED de la conexión Ethernet.
2. **DHCP:** La configuración mediante DHCP debe ser posible en todos los dispositivos. La descripción de este protocolo se puede encontrar en el Apartado 2.3.5, “DHCP”. El dispositivo debe soportar las opciones de DHCP de máscara de subred y router. La cámara enviará un DHCPDISCOVER y esperará a DHCPOFFER. Después enviará un

DHCPREQUEST y esperará respuesta con un DHCPACK. La cámara envía estos mensajes en intervalos de 2, 4 y 6 segundos en caso de que no vayan siendo respondidos. De esta forma, al pasar aproximadamente 12 segundos sin respuesta, se supondrá que no hay servidor DHCP presente. Las direcciones otorgadas por DHCP expiran tras un tiempo. En ese caso, la cámara debe dejar de usar esa dirección e iniciar de nuevo la configuración IP mediante DHCP.

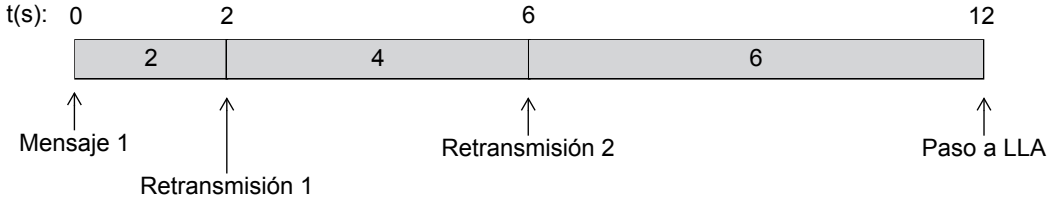


**Figura 3.5: Esquema de retransmisiones de los mensajes DHCP.**

3. **Dirección de Enlace Local** (*Link-Local Address*): Su implementación es obligatoria en todos los dispositivos. Las direcciones de enlace local son direcciones IP creadas únicamente para comunicaciones dentro de una subred local. Los routers no enrutan paquetes con direcciones de enlace local. Las direcciones IP tipo LLA se mapean en el rango 169.254.xxx.xxx. El dispositivo escoge una dirección de este rango y mediante ARP chequea sin se encuentra en uso.

El proceso de configuración IP garantiza que en aproximadamente 20 segundos como máximo la cámara dispone de un direccionamiento correcto.

### 3.2.1.2. Enumeración de dispositivos

Una vez que ha terminado la configuración IP con unos parámetros válidos, el dispositivo debe responder a las preguntas de descubrimiento de dispositivo procedentes de cualquier aplicación.

Mediante esta enumeración, las aplicaciones pueden saber qué dispositivos están conectados a la misma subred y averiguar parámetros del dispositivo tales como fabricante, modelo o dirección MAC.

Los mensajes de descubrimiento de dispositivos son parte del protocolo GVCP que se verá con detenimiento en el Apartado 3.3, “Control del dispositivo (GVCP)”.

Dependiendo de la difusión que tenga el mensaje de descubrimiento distinguimos:

a) **Descubrimiento de dispositivos broadcast:** Permiten a la aplicación buscar dispositivos que residen en la misma subred (no cruza routers). En el campo de IP destino se especifica la dirección 255.255.255.255 (dirección broadcast). Cualquier dispositivo que reciba este mensaje y disponga de una IP válida, debe responder con un mensaje unicast hacia la aplicación.

b) **Descubrimiento de dispositivo unicast:** Solo puede emplearse cuando la dirección IP del dispositivo es conocida de antemano por la aplicación. Se lleva a cabo enviando el mensaje directamente a la IP del dispositivo. Este a su vez debe responder con un mensaje unicast hacia la aplicación.

Normalmente la cámara tendrá un número de serie y su identificación en la carcasa. De esta forma, podremos contrastarlo con lo que nos devuelve el mensaje de descubrimiento.

### 3.2.2. Conexión y desconexión del dispositivo

La aplicación que controla a un dispositivo puede percatarse de que este se ha conectado mediante sus mensajes DHCP o, si no los soporta, enviando de vez en cuando mensajes de descubrimiento. En cuanto a la desconexión, se sabrá que un dispositivo se ha desconectado de la aplicación si no obtenemos respuesta a los mensajes que le enviamos.

## 3.3. Control del dispositivo (GVCP)

GVCP (*GigE Vision Control Protocol*) es un protocolo del nivel de aplicación que se asienta sobre el protocolo de transporte UDP. Las funciones básicas que cumple son:

- **Configurar el dispositivo** desde la aplicación.
- **Instanciar desde la aplicación canales de streaming** para el envío de imágenes desde el dispositivo.
- Permitir al dispositivo **notificar a la aplicación cuando eventos específicos acontecen**.

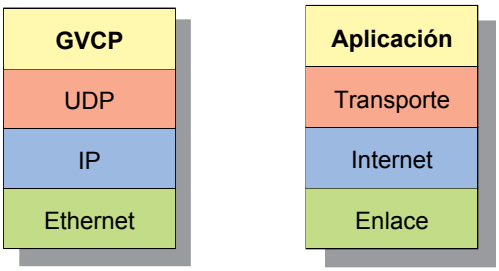


**Figura 3.6: Ubicación de GVCP en la pila de protocolos UDP/IP y en la pila OSI.**

Bajo GVCP, la aplicación actúa como *maestra* y la cámara como *esclava*. Los comandos siempre son iniciados por la aplicación. La cámara puede informar de su estado mediante mensajes en sentido contrario (no se consideran comandos en sí).

GVCP otorga el control a una sola aplicación. Es esta aplicación la que tendrá autorización para enviar los comandos de control. Sin embargo, es posible que otras aplicaciones monitoricen el dispositivo (lean el estado de sus registros) siempre que la aplicación principal lo permita.

Tanto los comandos enviados como las respuestas están contenidos en paquetes únicos (no fragmentados). Al correr sobre UDP, no tenemos garantizada la fiabilidad de que los mensajes se intercambien correctamente, por lo que GVCP define mecanismos para garantizar la correcta transmisión. El puerto de la cámara al que nos tenemos que referir es el 3956. Sin embargo, en el lado de la aplicación no existe ningún puerto concreto, por lo que se puede emplear cualquiera que esté libre.

La mayor parte del control de un dispositivo se hace en base a la escritura en los registros situados en su memoria. Estos registros se manejan mediante los comandos de control *ReadReg* y *WriteReg* que se verán en el Apartado 3.3.2.2, “Tramas del canal de control”. A su vez, una explicación más amplia de los registros se puede encontrar en el Apartado 3.5, “Registros”.

### 3.3.1. El concepto de canal

En GigE Vision, la información fluye a través de conexiones denominadas *canales*.

Los **canales** son enlaces virtuales empleados para transmitir información entre entidades de GigE Vision. El estándar da soporte a tres tipos de canales:

- **Canal de control:** Se emplea para comunicar la aplicación con el dispositivo. Siempre debe existir un canal de control para utilizar la cámara.
- **Canal de streaming:** Se abre cuando se desean transmitir imágenes. Se permite abrir hasta 512 canales de este tipo.
- **Canal de mensajes:** Permite a la cámara comunicar eventos que le acontecen. Se pueden instanciar de 0 a 1 canales de este tipo.

Todos los canales emplean la misma interfaz de red. Para diferenciar los datos de unos u otros, se asigna a cada uno un puerto UDP distinto. En el lado de la cámara se ha definido un puerto GVCP estándar para el canal de control (3956). Para el resto, se puede usar cualquier puerto disponible. Los canales se crean dinámicamente y pueden abrirse o cerrarse por parte de la aplicación cuando se desee.

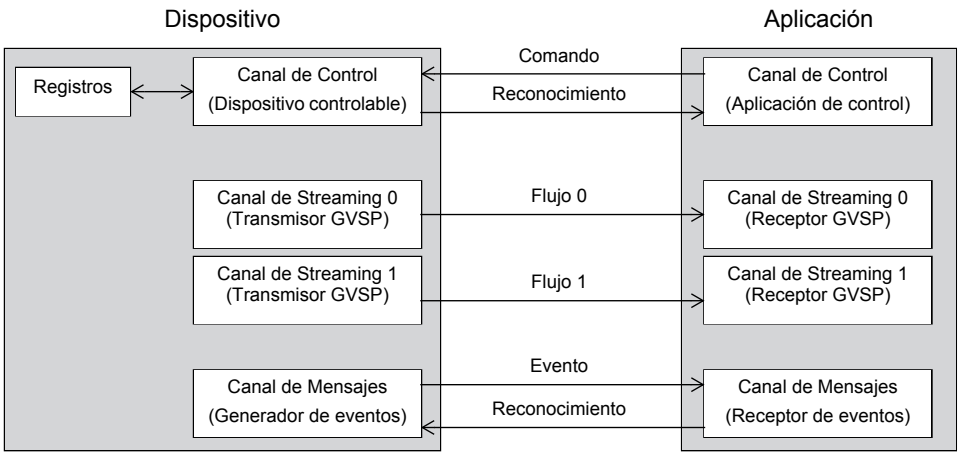


**Figura 3.7: Ejemplo básico de canales establecidos entre un dispositivo y una aplicación.**

#### 3.3.1.1. Canal de control

El canal de control se emplea para que **la aplicación se comunique con el dispositivo**. Se definen dos tipos:

- **Canal de control primario:** Creado por la aplicación que queremos que tenga el control (aplicación primaria). Si tuviéramos varios dispositivos queriendo ejercer este control privilegiado, sólo el primero que lo solicitara sería el que lo consiguiera. La aplicación primaria es la única aplicación que tiene permitido escribir en los registros del dispositivo.
- **Canal de control secundario:** Creados por aplicaciones secundarias. Sólo pueden leer el estado de registros con el objetivo de monitorizar y/o depurar.

La existencia del **canal de control es obligatoria** y debe ser creado antes de cualquier canal de otro tipo.

La secuencia que siguen las peticiones en el canal de control se muestra en la Figura 3.8.

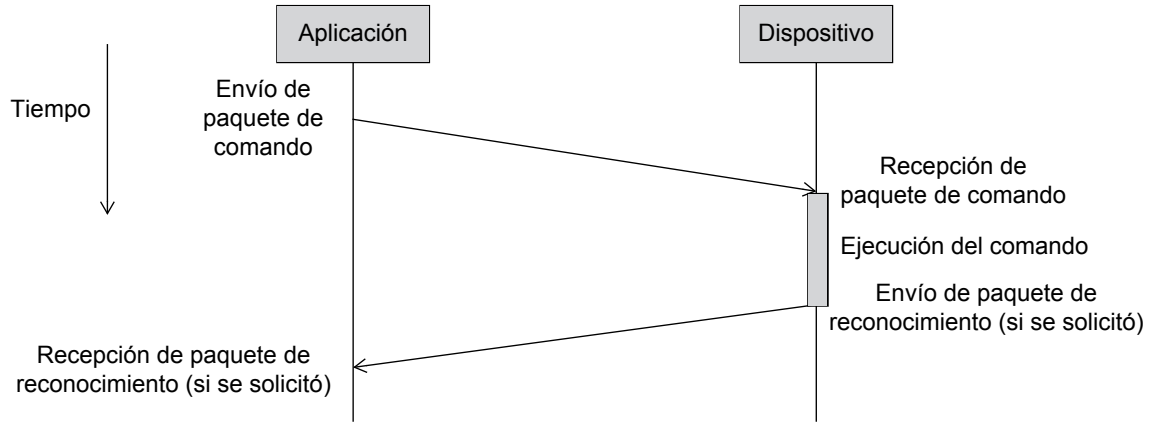


**Figura 3.8: Secuencia de un comando en el canal de control.**

Privilegios del canal de control

GVCP define cuatro niveles de privilegio para las aplicaciones respecto al dispositivo. Sólo el primer nivel es soportado por cualquier cámara GigE Vision. El resto son opcionales:

1. **Acceso Exclusivo:** La aplicación primaria controla el dispositivo (puede leer y escribir en él). Ninguna otra aplicación puede tener acceso. En caso de que lo intentara, el dispositivo retorna que el acceso está denegado.
2. **Control de acceso:** Hay una aplicación primaria, pero las otras pueden leer del dispositivo.
3. **Control de acceso con conmutación activada:** Similar al nivel 2, salvo que en este caso otra aplicación puede tomar el control si lo desea.
4. **Acceso de monitorización:** La aplicación puede leer del dispositivo si no existe ninguna aplicación primaria.

Para que el dispositivo sepa en que contexto se mueve, debe memorizar la siguiente información:

- IP de la aplicación.
- Puerto de origen de la aplicación.
- El privilegio asociado.

Apertura y cierre del canal de control. Control del dispositivo.

Para abrir o cerrar un canal de control, debemos escribir en un registro de la cámara. Esta nos responderá afirmativa o negativamente en función de si contamos con acceso o no.

Una vez la aplicación ha abierto un canal de control primario, puede enviar cualquier comando GVCP. Las aplicaciones secundarias pueden enviar los comandos de lectura de registros y memoria y cualquier aplicación puede enviar un comando de descubrimiento.

Heartbeat del canal de control

En el posible caso de que uno de los dos participantes en la comunicación sea desconectado abruptamente, GVCP ha implementado un mecanismo que lo detecta.

Heartbeat (en inglés, latido de corazón) significa que la aplicación debe manifestar frente al dispositivo cada cierto tiempo que se encuentra ahí. El dispositivo dispone de un contador que si se agota hace que se cierre el canal de control. Cualquier comando enviado por la aplicación primaria reinicia este contador. En consecuencia, se debe tener en cuenta que si no se están enviando comandos GVCP cada cierto tiempo, la aplicación deberá enviar alguno aunque no lo considere necesario. Normalmente se suele enviar algún comando que lea un registro.

Fiabilidad y recuperación de errores

Como ya se ha comentado, UDP no permite verificar si la entrega de los paquetes se ha realizado con éxito, pudiendo perderse de forma que pasa inadvertida a la aplicación. Se precisa saber si cada comando que enviamos ha llegado correctamente y cual es su resultado. Para ello, cada mensaje puede ser enviado con la opción de que un mensaje en sentido contrario sea enviado como reconocimiento. Para saber qué reconocimiento corresponde a qué mensaje enviado, en los paquetes enviados al dispositivo se especifica un campo *req_id* y en los recibidos del dispositivo un campo *ack_id*. En la Figura 3.9 se muestra un esquema de su funcionamiento:

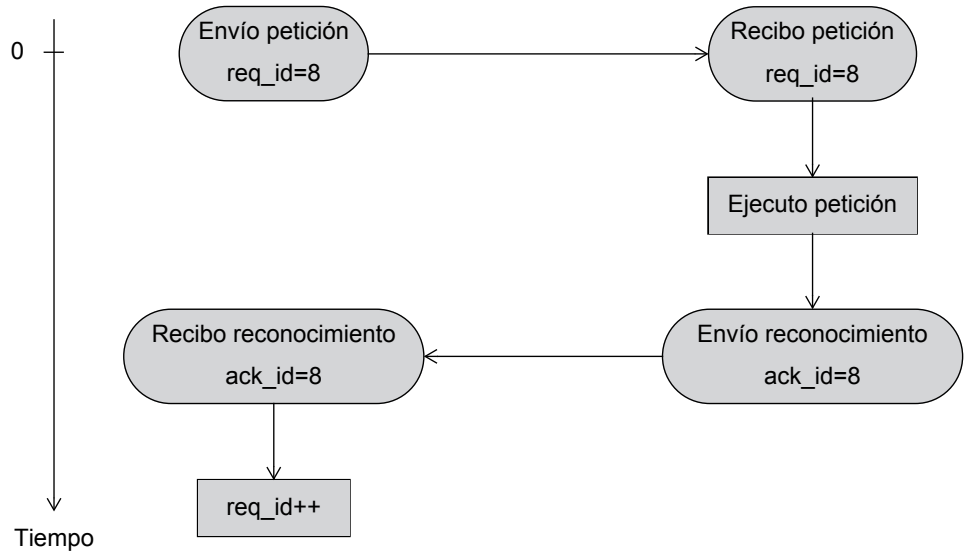


**Figura 3.9: Método de envío de comando y recepción de su reconocimiento.**

En el caso de que no recibamos repuesta, el estándar recomienda que el mensaje sea enviado hasta en 3 ocasiones. Si se agotan, la aplicación debe avisar al usuario.

Si se da el caso de que hayamos requerido dos veces seguidas el mismo comando y el dispositivo haya recibido ambas, no lo ejecuta por dos veces sino que con una de ellas es suficiente. Sin embargo, si que enviará por dos veces el reconocimiento.

Por otro lado, como es sabido los niveles inferiores de la pila de protocolos UDP, IP y Ethernet disponen de CRC. En GigE Vision, el CRC de UDP es opcional. Sin embargo, la no coincidencia del CRC de IP o Ethernet supone el descarte de la trama en el dispositivo.

Procesamiento lento de comandos

En ocasiones, la ejecución de un comando por parte del dispositivo conlleva más tiempo que el que la aplicación espera. En ese caso, existe un mensaje denominado PENDING_ACK que la cámara envía para informar de que va a tardar más de lo esperado.

### 3.3.1.2. Canal de streaming

Permite **transferir datos desde un transmisor a un receptor empleando el protocolo GVSP** (se verá en el Apartado 3.4, "Obtención de imágenes (GVSP)"). Los canales de streaming pueden transmitir distintos tipos de datos. Lo más común es que se utilicen para transmitir imágenes, pero también pueden transmitir otro tipo de información como histogramas, datos estadísticos, etc.

Los canales de streaming son unidireccionales. Esto significa que los extremos pueden ser únicamente fuente o receptor, pero no ambos a la vez.

Apertura y cierre de un canal de streaming

Para abrir un canal de streaming solo hay que escribir utilizando el comando apropiado en el registro del dispositivo que almacena la IP del host donde queremos que se envíen los datos. El cierre se produce escribiendo 0 en el puerto.

Abrir un canal de streaming no implica que se vayan a recibir los datos de forma inmediata. Aunque en el estándar no se especifica, la experiencia con la cámara en el proyecto ha hecho ver que es necesario escribir en un registro específico para lanzar la captura de imagen.

### 3.3.1.3. Canal de mensajes

El canal de mensajes permite al dispositivo **enviar mensajes asíncronos a la aplicación**. Por ejemplo, que la cámara quiera señalizar que se ha cumplido una determinada condición y ha saltado un disparador. Es un mecanismo que se da por si existiera esa necesidad de comunicación. Sin embargo, no es una característica obligatoria.

El canal de mensajes es bastante similar al canal de control, salvo porque los mensaje se envían en sentido opuesto. Las cabeceras empleadas son idénticas y también implementa el mecanismo de fiabilidad en base al req_id.

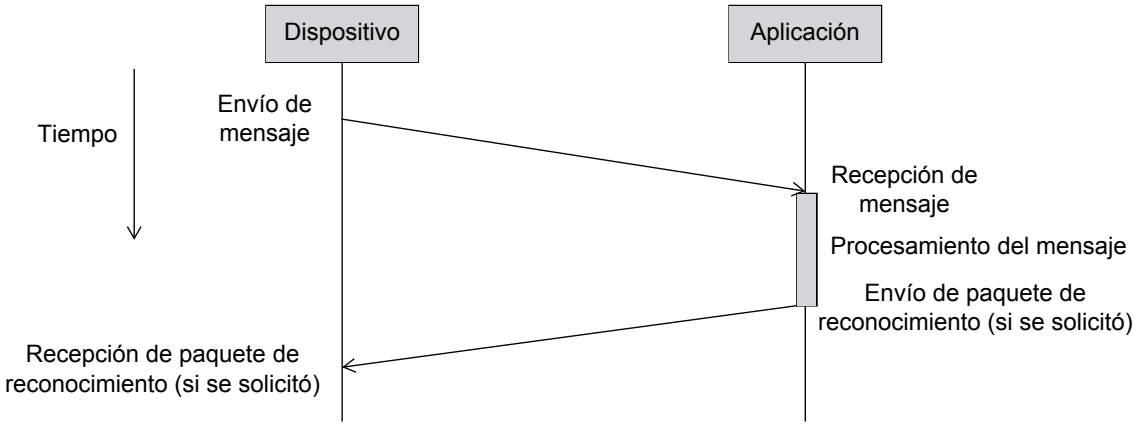


**Figura 3.10: Secuencia de un mensaje en el canal de mensajes.**

Existen dos categorías de eventos asíncronos. La primera es la de los eventos definidos en el propio estándar GigE Vision. La segunda es la de los eventos definidos por el fabricante para el dispositivo.

Apertura y cierre del canal de mensajes

Escribiendo en los registros del dispositivo la IP y el puerto del computador que recibirá los mensajes, se puede abrir el canal de mensajes. Dejando a 0 el puerto anteriormente escrito se cierra el canal.

## 3.3.2. Formato de las tramas

Las tramas GVCP tienen un tamaño múltiplo de 32 bits. El orden de secuenciación de bytes es big-endian (bytes menos significativos antes que los más significativos), es decir, el orden empleado por los protocolos Ethernet, IP y UDP.

## 3.3.2.1. Cabeceras

El protocolo GVCP define dos tipos de cabeceras. Una para los comandos y otra para los mensajes de reconocimiento. Estas cabeceras son comunes a todos los mensajes GVCP. La particularidad de cada mensaje viene al añadir a estas cabeceras otros campos propios de cada uno.

Cabecera de comando

La Figura 3.11 muestra el formato de la cabecera de comando.

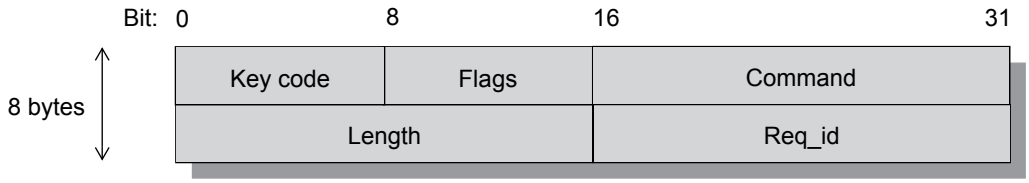


**Figura 3.11: Cabecera de comando.**

Los campos tienen la siguiente función:

- **Key code** [1 byte]: Código empleado para facilitar la identificación de los mensajes GVCP. Su valor siempre es 0x42.
- **Flags** [1 byte]: Su flag más importante es *acknowledge*. Si está activada significa que el dispositivo debe enviar un mensaje de reconocimiento.
- **Command** [2 bytes]: Código para identificar el comando a ejecutar.
- **Length** [2 bytes]: Número de bytes contenidos en el mensaje sin incluir esta cabecera.
- **Req_Id** [2 bytes]: Valor empleado para asociar a un comando con su respuesta. Generado por la aplicación, el dispositivo copia este valor en el campo ack_id del mensaje de respuesta.

Cabecera de reconocimiento

La Figura 3.12 muestra el formato de la cabecera de reconocimiento.

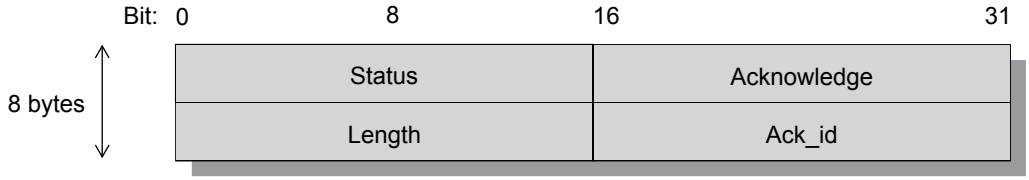


**Figura 3.12: Cabecera de reconocimiento.**

Los campos tienen la siguiente función:

- **Status** [2 bytes]: Estado de la operación ejecutada. Nos informa de si el resultado es exitoso o se ha producido algún error.
- **Acknowledge** [2 bytes]: Código que identifica el comando ejecutado. No es el mismo que el del campo Command de la cabecera de comando, pues se usa un código distinto para el envío y para la respuesta.
- **Length** [2 bytes]: Número de bytes contenidos en el mensaje sin incluir esta cabecera.
- **Ack_Id** [2 bytes]: Valor empleado para asociar una respuesta al comando que la generó. Tiene el mismo valor que el req_id del comando que la generó.

### 3.3.2.2. Tramas del canal de control

Los mensajes de esta sección son enviados a través del canal de control desde la aplicación al dispositivo y las respuestas de reconocimiento circulan en sentido contrario. Antes de ejecutar cualquier comando, el dispositivo verifica que el solicitante cuenta con el privilegio necesario.

Las tramas del canal de control llevan las cabeceras definidas en el apartado anterior. En ocasiones no son necesarias más que esas cabeceras para cumplir el objetivo del mensaje. Sin embargo, es común que a los campos de la cabecera se añadan otros específicos en el caso de mensajes que transportan información adicional.

No se va a hacer una descripción detallada de los mensajes que a continuación se muestran. Esta memoria pretende únicamente dar una visión de los mensajes que existen para comprender el funcionamiento general. El formato concreto de las tramas y sus campos puede encontrarse en el estándar.

**Discovery**

Comando para **enumerar los dispositivos**. No se requieren privilegios para ejecutarlo.

Tramas:

- **Discovery_Cmd:** Petición para enumerar dispositivos.
- **Discovery_Ack:** Respuesta a la petición de enumeración. Cada dispositivo presente debe enviar este mensaje que incluye entre otros datos la configuración de red asignada al dispositivo, el modelo y nombre de fabricante, etc.

**ForceIP**

En algunas ocasiones puede ser útil para la aplicación “forzar” una dirección IP en el dispositivo. Por ejemplo, cuando un dispositivo está usando una IP persistente que no conocemos, puede ser difícil para la aplicación comunicarse con él si la red a la que pertenece es distinta a la que ocupa la aplicación. Este problema no existe cuando la asignación de dirección se realiza con DHCP o LLA, ya que en estos casos la IP es negociada.

Para solventar este problema, GVCP proporciona un mecanismo para **forzar una IP estática en el dispositivo** siempre que la aplicación sea conocedora de la dirección física (MAC) del mismo.

Otra función de ForceIP es **reiniciar el proceso de obtención de dirección** visto en el Apartado 3.2.1.1, “Configuración IP”.

Tramas:

- **ForceIP_Cmd:** Este mensaje tiene un campo en el que se especifica la MAC del dispositivo. El dispositivo cuya MAC coincida con la del mensaje debe establecer sus parámetros de direccionamiento al valor también especificado en este mensaje o, en el caso de que no hayan sido incluidos, reiniciar el proceso de obtención de dirección.
- **ForceIP_Ack:** Enviado por el dispositivo si una dirección IP estática ha sido asignada.

### ReadReg

La aplicación puede enviar un mensaje ReadReg para **leer un registro de dispositivo**. El estándar da la posibilidad de que se concatenen varias lecturas en un solo mensaje hasta un máximo de 135 registros. Las lecturas múltiples no son soportadas por todos los dispositivos. Los registros son de 32 bits.

Tramas:

- **ReadReg_Cmd:** Comando para la lectura de registros. Se especifican las direcciones de lo que se pretende leer.
- **ReadReg_Ack:** Datos de los registros pedidos. Como se podían haber pedido uno o varios contenidos de registros, el campo *length* de la cabecera nos indica la cantidad de datos transmitida.

### WriteReg

Mensaje para **escribir en registros del dispositivo**. Al igual que en la lectura, se pueden concatenar varias escrituras en un solo mensaje con un máximo de 67 registros. La diferencia de número frente a los 135 registros leídos en ReadReg está en que en este caso se necesita el doble de espacio ya que aparte de la dirección del registro a escribir hay que adjuntar el dato que se desea modificar. Las escrituras múltiples no son soportadas por todos los dispositivos. Los registros son de 32 bits.

Tramas:

- **WriteReg_Cmd:** Mensaje para la escritura de uno o varios registros. Se especifican de forma alternada la dirección y el dato que va en cada registro.
- **WriteReg_Ack:** Devuelve el número de operaciones de escritura realizadas correctamente.

### ReadMem

Comando que permite la **lectura en el dispositivo de localizaciones de memoria** consecutivas de 1 byte.

Tramas:

- **ReadMem_Cmd:** Petición de lectura de la memoria del dispositivo en la que se especifica la dirección base y el número de bytes de información que se desea leer.
- **ReadMem_Ack:** Mensaje de respuesta que contiene la dirección base desde la que se comenzó a leer y los datos leídos. El campo *length* de la cabecera nos recuerda el número de bytes leídos.

### WriteMem

Comando cuya implementación en los dispositivos es opcional y que permite **escribir datos en la memoria**.

Tramas:

- **WriteMem_Cmd:** Mensaje para solicitar escritura en la memoria. Se especifica la dirección base y los datos a escribir.
- **WriteMem_Ack:** Respuesta al mensaje de escritura en memoria en el que se incluye el número de bytes satisfactoriamente escritos.

### PacketResend

El comando PacketResend se utiliza (en el caso de que el dispositivo lo soporte) para solicitar la retransmisión de paquetes perdidos. Se diferencia del resto de comandos en que es asíncrono, pues no sigue el esquema de esperar al reconocimiento del mensaje anterior. De hecho, su esencia es esa, cuando esperamos un tiempo prudencial y no llega un mensaje de respuesta, se emplea PacketResend para solicitar dicha respuesta por si el comando no había sido ejecutado o la respuesta se perdió por el camino. No hace falta solicitar reenvíos de un solo reconocimiento sino que se pueden solicitar en bloques.

Otras peculiaridades de este comando es que la aplicación no tiene que preocuparse de seguir la secuencia de req_id como ocurre con otros comandos GVCP, puede usar cualquiera. Para lo que sí usa el código req_id es para identificar los comandos que se necesita reenviar. Este comando no se puede emplear para resetear el heartbeat del canal de control.

Tramas:

- **PacketResend_Cmd:** Mensaje para solicitar reconocimiento de comandos. Se especifica el req_id que se envió con el comando.
- **PacketResend_Ack:** No existe.

### Pending

Característica opcional que permite indicar que el procesamiento de un comando va a llevar más tiempo del esperado. El tiempo de espera hasta el envío de este mensaje se encuentra especificado en un registro de la cámara. El mensaje es enviado por la cámara en el momento en que presupone que la ejecución de un comando va a llevar más tiempo del especificado en dicho registro.

Tramas:

- **Pending_Ack:** Este mensaje incluye el número de ms que el dispositivo ha calculado tardará en completar la petición en curso.

**Action**

Comando de implementación opcional que permite desencadenar varias acciones en uno o varios dispositivos. A grandes rasgos, esta característica del estándar (no descrita en detalle en este resumen), permite mediante el empleo de máscaras e identificadores ejecutar los comandos que se deseen de los especificados anteriormente y en los dispositivos que definamos.

Tramas:

- **Action_Cmd:** Mensaje para desencadenar las acciones.
- **Action_Ack:** Respuesta enviada si se han cumplido las condiciones de ejecución.

### 3.3.2.3. Tramas del canal de mensajes

Las tramas de este tipo son enviadas a través del canal de mensajes (en el caso de que el dispositivo lo soporte). Es el dispositivo el que siempre inicia la transacción dentro del canal de mensajes.

Las tramas del canal de mensajes emplean las cabeceras definidas en el Apartado 3.3.2.2, "Tramas del canal de control", las mismas que para los comandos de control.

**Event**

Usadas por el dispositivo para **notificar que han ocurrido eventos asíncronos**. Varios eventos pueden ser concatenados en un único mensaje, con una limitación a 33 por el tamaño del mensaje. Los eventos se identifican mediante un código. Algunos de ellos son definidos por el estándar GigE Vision. El resto pueden ser usados para definir otros en la aplicación.

Tramas:

- **Event_Cmd:** Mensaje enviado por el dispositivo informando del evento o eventos acontecidos. Contiene para cada evento (entre otros datos) un identificador o el momento en que se produjo.
- **Event_Ack:** El dispositivo no precisa obligatoriamente de un mensaje de reconocimiento por parte de la aplicación. En el caso de que se necesitara, el mensaje contendría sólo la cabecera de reconocimiento.

**EventData**

El dispositivo emplea EventData para **notificar que han ocurrido eventos asíncronos**. La principal diferencia con Event es que **permite añadir datos al mensaje** y que sólo se permite un único evento por mensaje.

Tramas:

- **EventData_Cmd:** Mensaje enviado por el dispositivo informando del evento acontecido. Contiene entre otros datos un identificador o el momento en que se produjo. Además, lleva asociada una sección en la que se pueden adjuntar datos.

- **EventData_Ack:** El dispositivo no precisa obligatoriamente de un mensaje de reconocimiento por parte de la aplicación. En el caso de que se necesitara, el mensaje contendría sólo la cabecera de reconocimiento.

### 3.3.3. Fichero XML de configuración del dispositivo

Los dispositivos GigE Vision deben disponer de un fichero XML que describa las capacidades con las que cuenta: desde comandos soportados a formatos de imagen manejados. El fichero debe seguir la sintaxis descrita en el estándar *GenICam* que define las interfaces que deben seguir las cámaras. Dado que el fichero puede llegar a ser bastante grande (del orden de varios cientos de kilobytes) puede ser encontrado en formato comprimido *.ZIP*.

Tres son los lugares donde podremos encontrar este fichero:

- Memoria no volátil del dispositivo.
- Sitio web del fabricante.
- En un directorio de la máquina que corre la aplicación: El fichero vendrá normalmente en un CD con el dispositivo. El usuario debe copiar este fichero a un directorio predefinido desde el cual la aplicación pueda tener acceso.

Dos registros del dispositivo nos indican la localización de entre estas tres posibles.

## 3.4. Obtención de imágenes (GVSP)

GVSP (*GigE Vision Streaming Protocol*) es un protocolo del nivel de aplicación que corre sobre la capa de transporte UDP. Permite a un receptor GVSP la obtención de imágenes, datos sobre las imágenes y otra información que le proporcione un transmisor GVSP. En definitiva, se trata de un protocolo para el transporte de información sobre imágenes (información de cualquier tipo) entre un emisor y un receptor GVSP. Sin embargo, lo más habitual es el transporte de imágenes en sí.

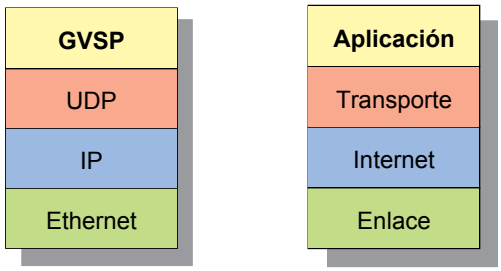


**Figura 3.13: Ubicación de GVSP en la pila de protocolos UDP/IP y en la pila OSI.**

Los paquetes GVSP siempre viajan desde el transmisor al receptor, sin que estos varíen su papel en la comunicación.

Debido a que GVSP se asienta sobre UDP e IPv4, que son protocolos no fiables, GVSP puede proveer mecanismos que permitan garantizar la fiabilidad en la transmisión de los paquetes.

Los objetivos del GigE Vision Streaming Protocol (GVSP) son los siguientes:

- Definir un **mecanismo para que un transmisor GVSP pueda enviar imágenes**, datos sobre imágenes u otros datos.
- **Definir descripciones de los datos** que transporta.
- **Minimizar la complejidad de transmisión** y el ancho de banda empleado.

Otras consideraciones generales sobre el protocolo de transporte GVSP son las siguientes:

- **Requerimientos del tamaño del paquete:** Contrariamente a como ocurría con los paquetes GVCP, el tamaño de los paquetes GVSP no tiene que ser múltiplo de 32 bits. La resolución es de bytes, es decir, múltiplo de 8 bits.
- **Fiabilidad y recuperación de errores:** En el caso de que un paquete se pierda, se puede solicitar el reenvío mediante un el comando PacketResend de GVCP.
- **Control de flujo:** Al no existir mensaje de reconocimiento, la espera hasta suponer que se ha producido una pérdida del paquete se realiza en base a un tiempo entre mensajes especificado en un registro de la cámara. Así, si se produce una espera mayor que este tiempo, se podría solicitar el reenvío.
- **Conexión punto a punto:** La pérdida de la conexión es detectada mediante el mecanismo de heartbeat de GVCP. Si la aplicación es desconectada, el temporizador de heartbeat de GVCP expirará y automáticamente reseteará la conexión de streaming.

### 3.4.1. Bloques de datos

La información volcada en el canal de streaming se divide en *bloques* y a su vez estos bloques se dividen en paquetes Ethernet. Una cámara GigE Vision hará corresponder cada imagen capturada con un bloque de datos. Por su parte, el sistema receptor será capaz de identificar cada imagen observando el identificador asociado a cada bloque. Los bloques de datos contienen tres elementos:

1. **Data Leader:** Un único paquete que señaliza el comiendo de un nuevo bloque de datos.
2. **Data Payload:** Uno o más paquetes conteniendo la información que ha ser transmitida dentro de un bloque de datos. Ejemplo: una imagen.
3. **Data Trailer:** Único paquete que señaliza el final de un bloque de datos.

Un ejemplo de bloque de datos típico es el de la Figura 3.14.

Los bloques de datos son enviados por el dispositivo de forma secuencial en un canal de streaming.

Para transportar la información de manera eficiente, GVSP define varios tipos de datos a los que corresponde la definición de una trama distinta:

1. **Image:** En este formato se manejan imágenes codificadas como matrices de pixels. El orden en que se transmiten los pixels de la imagen es de izquierda a derecha y después de arriba a abajo. La posición que en la imagen ocupa el primer pixel del paquete puede ser obtenida multiplicando *packet_id*-1 (packet_id es un campo de la cabecera GVSP que se verá más adelante) por el tamaño del paquete.
2. **Raw data:** Se emplea para hacer streaming de datos sin formato (datos en crudo). Un ejemplo es el envío de estadísticas de adquisición.
3. **File:** Usado para transmitir ficheros que pueden ser directamente almacenados en el disco duro del receptor. Por ejemplo, un transmisor puede enviar los datos en compresión JPEG.
4. **Chunk data:** Formato empleado para enviar porciones de datos con etiquetas. Cada porción se compone de los datos y de una etiqueta. La etiqueta a su vez cuenta con un identificador de la porción y un campo con la longitud de los datos transmitidos.

5. **Extended chunk data:** El formato Chunk Data resulta ambiguo cuando se desea reconstruir la imagen que se recibe. Este formato añade más información para que esta tarea sea más fácil.
6. **Device-specific:** Empleado para transmitir información específica del dispositivo.

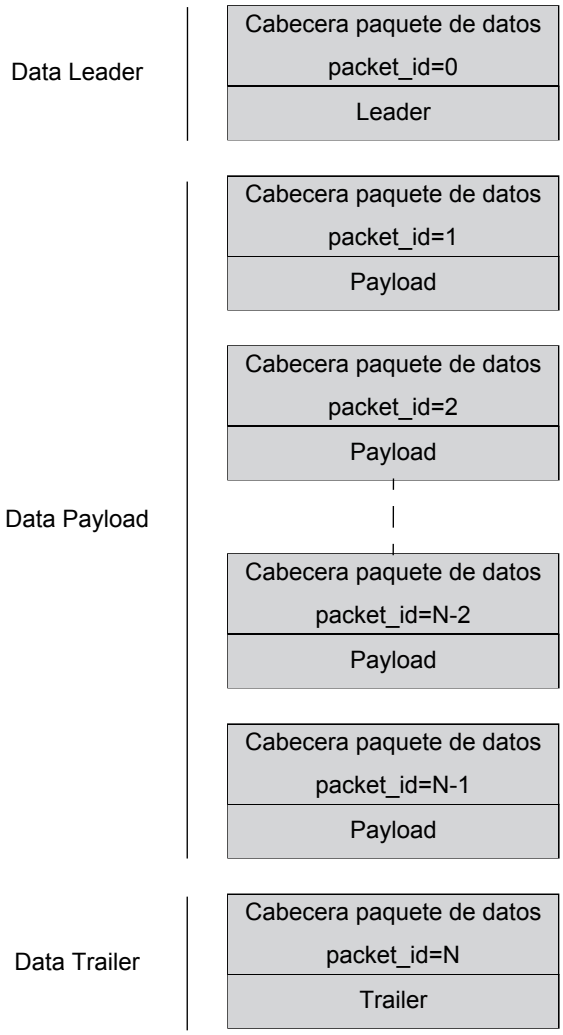


**Figura 3.14: Bloque de datos.**

## 3.4.2. Formato de las tramas

El orden de secuenciación de bytes que se sigue es *big-endian*.

### 3.4.2.1. Cabecera de datos

El protocolo GVSP define un único tipo de cabecera que deben llevar todos los paquetes GVSP. A esta cabecera se le añaden los campos necesarios en cada subcabecera específica si los necesitara.

Las tramas GVSP se estructuran en tres grupos dependiendo de si son destinadas a Data Leader, Data Payload o Data Trailer. Dentro de cada uno de estos grupos, se definen otras 6 tramas correspondientes a cada tipo de datos manejado.

La Figura 3.15 muestra el formato de la cabecera de única de GVSP.

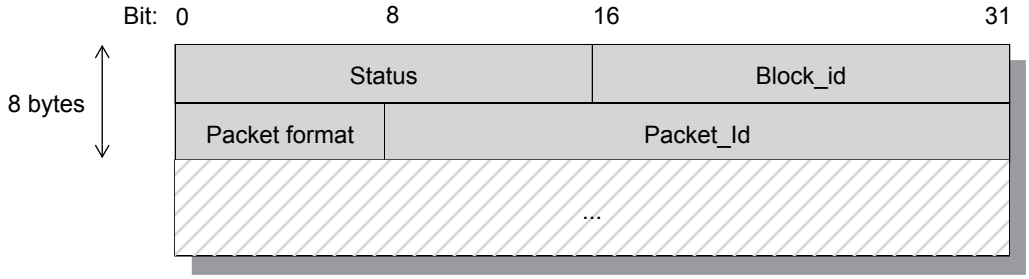


**Figura 3.15: Cabecera GVSP.**

Los campos tienen la siguiente función:

- **Status** [2 bytes]: Estado de la operación de streaming.
- **Block_id** [2 bytes]: Identificador del bloque de datos. Todos los datos de, por ejemplo una imagen, llevan el mismo.
- **Packet format** [1 byte]: Indica si el paquete contiene un Data Leader, Data Payload o Data Trailer.
- **Packet_Id** [3 bytes]: Identificador del paquete dentro del bloque. Permite saber el orden que llevan los trozos de datos dentro del bloque.

### 3.4.2.2. Tramas del canal de streaming

Los campos de estas tramas van anexos a los de la cabecera explicada en el apartado anterior.

Distinguimos tres tipos:

- **Data Leader:** Es el primer paquete del bloque enviado por la cámara. Cuenta con un campo en el que se especifica el tipo de bloque de datos a recibir. Además, se adjunta el instante de tiempo en que se generó el bloque.
- **Data Payload:** Transportan únicamente la información que se desea transmitir, sin añadir campos adicionales a la trama. Todos los paquetes de un mismo bloque deben ser del mismo tamaño salvo el último, que puede ser de tamaño menor.
- **Data Trailer:** El Data Trailer debe ser el último paquete del bloque. Una vez este mensaje es enviado, el dispositivo incrementa el identificador de bloque *block_id* y pone a 0 el identificador de paquete *packet_id*. El único campo adicional respecto de la cabecera general GVSP contiene el tipo de datos transportado.

A su vez, como ya se ha comentado, dentro de estos tres tipos se definen 6 tramas nuevas (dando un total de 18) que tienen ciertas particularidades en función del tipo de datos que transporten. Los tipos de datos/tramas son los ya mencionados con anterioridad: Image Data, Raw Data, File Data, Chunk Data, Extended Chunk Data y Device-specific Data.

El único formato que se empleará en este proyecto es Image Data. De los tres tipos de tramas referentes a Image Data (Image Data Leader, Image Data Payload e Image Data Trailer), es la Image Data Leader la que merece alguna explicación adicional.

Image Data Leader cuenta con campos en los que se informa del formato de pixel de los datos enviados así como las dimensiones de la imagen y de la distancia de la parte superior izquierda de la imagen respecto al origen de píxeles de la cámara. Estos datos son los que permiten implementar la denominada región de interés (ROI).

### 3.4.3. Formatos de pixel

GigE Vision soporta diversos formatos de imagen y pixel. Para describirlos, se señalan las siguientes características:

- **Formato de imagen:** Incluyen Mono (blanco y negro), RGB (color por componentes roja, verde y azul), varios tipos de Bayer (popular formato digital de color que también usa el rojo, verde y azul) y varios tipos de YUV (formato por componentes denominadas luminancia y crominancia). En el estándar también se define el orden que ocupa cada componente en el espacio de memoria destinado a cada pixel.
- **Alineación de pixel:** Cualquier formato de imagen que emplee más de un byte por componente de color del pixel empleará little-endian. El estándar también da la posibilidad de que los dispositivos permitan usar el formato big-endian.

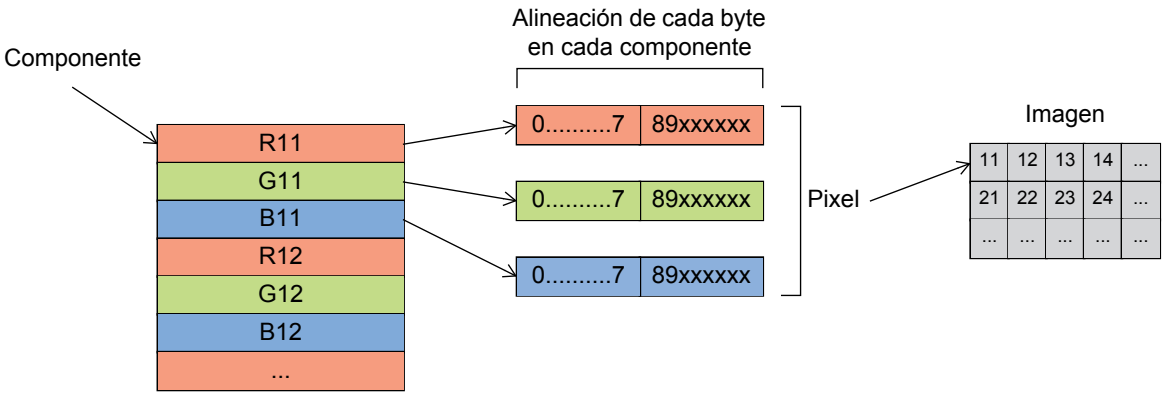


**Figura 3.16: Formato de imagen y alineación de pixel para RGB10_PACKED.**

## 3.5. Registros

Las cámaras GigE Vision cuentan con una memoria cuyo contenido ha sido dividido en zonas en las que se almacenan diversos datos y características de la cámara. Dependiendo de su longitud, pueden ser accedidas mediante los comandos de lectura/escritura registros (para 32 bits) o comandos de lectura/escritura de memoria (para accesos a localizaciones de tamaño distinto a 4 bytes).

Los registros son la piedra angular del funcionamiento de la cámara. Es de ellos de donde el dispositivo puede conocer qué es lo que debe de hacer y plasmar su estado. Prácticamente todos

los mensajes vistos hasta ahora emplean la escritura o lectura de registros de forma explícita (ReadReg, WriteReg, ReadMem, WriteMem) o implícita (por ejemplo, la cámara escribe su configuración IP tras realizar el proceso DHCP).

El estándar GigE Vision define los registros/segmentos de las primeras direcciones de memoria. Algunos de ellos contienen datos como la dirección MAC del dispositivo, la IP utilizada, la velocidad del enlace, los privilegios sobre la cámara, etc.

A continuación de los definidos por el estándar se encuentran los registros específicos de cada dispositivo, los cuales dependen de la especificación del fabricante.

## 3.6. Ejemplo mínimo de utilización de los comandos

Para finalizar este capítulo, se pretende mostrar un breve y esquemático ejemplo de cómo se podría dar uso a los mensajes explicados en estos apartados. En el ejemplo, se busca inicializar una cámara recién conectada y recibir imágenes de ella.

Este proceso comienza con la **fase de descubrimiento** que consta de:

1. El **negociado DHCP**
2. Envío de **comando de descubrimiento** (Discovery_Cmd) y recepción de respuesta (Discovery_Ack).

A esta fase le sigue otra en que **se abren los canales necesarios** para recibir imágenes:

1. **Apertura de canal de control** escribiendo los permisos de la aplicación en un registro concreto de la cámara mediante el comando Writereg_Cmd. No es obligatorio recibir confirmación.
2. **Apertura del canal de streaming** escribiendo la IP donde queremos que se reciban los datos mediante un comando Writereg_Cmd. No es necesario recibir confirmación.

Estas son las fases que describe el estándar para alcanzar el punto en que se está en condiciones de recibir imágenes, las cuales llegan en paquetes de tipo GVSP.

# 4. Implementación

## 4.1. Visión general

En este capítulo se pretende mostrar la forma en que han sido abordados los retos teóricos planteados en capítulos anteriores. La Figura 4.1 muestra de forma esquemática las capas en que se divide la lógica implementada. Se parte de un driver Ethernet que ya existe y que permite enviar y recibir las tramas intercambiadas con la cámara. El driver captura las tramas y las hace disponibles a las capas superiores sin hacer ningún tipo de procesado. Las tramas se contienen unas a otras. Según se asciende en el esquema, cada capa extrae la parte de los datos que le corresponde.

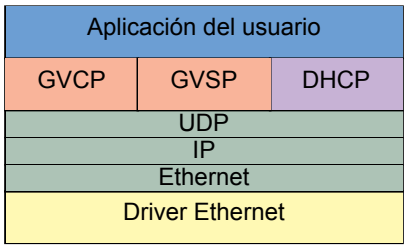


**Figura 4.1: Esquema de capas a implementar (salvo driver Ethernet que ya existe).**

Los datos manejados por el driver Ethernet se interpretan dentro de la pila Ethertet, IP y UDP. Esta pila se ha desarrollado de forma que constituye un resultado independiente de este proyecto que podría utilizarse en cualquier otra aplicación que requiera de estos protocolos.

Sobre esta pila se han implementado las tramas del estándar GigE Vision y también se ha dado soporte a DHCP, que constituye otro ejemplo de resultado independiente que se podrá usar en otras aplicaciones.

En el último nivel se encuentra el código de aplicación desarrollado para hacer uso de las tecnologías subyacentes de forma sencilla por parte del usuario.

El apilamiento de protocolos, encapsulando una trama dentro de otra, sigue el esquema que se pudo ver en la Figura 2.3.

# 4.2. Directrices de implementación

## 4.2.1. Driver Ethernet de MaRTE

El empleo en este proyecto de la interfaz Ethernet como mecanismo para la comunicación de los datos requiere de la existencia de un controlador o driver sobre MaRTE que proporcione las operaciones básicas para enviar y recibir datos.

MaRTE dispone de controladores para tarjetas de red Ethernet que emplean los chips *eepro100*, *rtl8139* o *sis900*. Estos controladores se encargan de la transmisión y recepción de datos en el nivel más cercano al hardware. Es necesario comprobar si la tarjeta de red de que se dispone en el equipo que ejecuta MaRTE cuenta con uno de estos chips, ya que de lo contrario no podremos hacer uso de ellas.

El driver Ethernet de MaRTE constituye un módulo software previo al desarrollo de este proyecto. Sin embargo, al ser empleado en este trabajo puede resultar interesante introducirse en el código para conocer alguna característica de implementación o modificar algún parámetro del mismo. No obstante, no es objetivo de este proyecto modificar el driver más allá de cambios menores. Por ejemplo en el proyecto se pudo, consultando el driver, conocer el tamaño del buffer de recepción de datos y modificar su valor dentro de los parámetros permitidos. Esto fue debido a que durante las pruebas realizadas al sistema, se podía ver que al aumentar la tasa de transmisión de datos por parte de la cámara el buffer tendía a desbordarse.

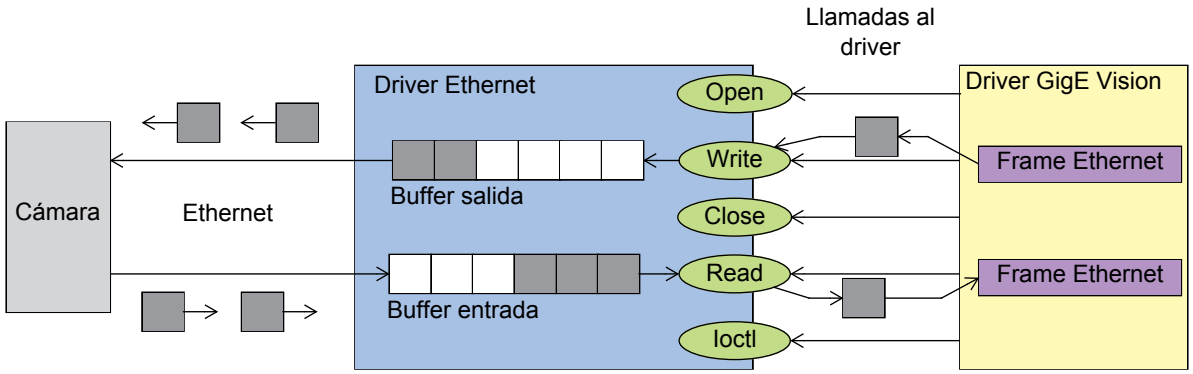


**Figura 4.2: Esquema de funcionamiento del driver Ethernet y del modo en que interaccionan con él la cámara y el driver GigE Vision.**

Siguiendo el modelo descrito en el estándar POSIX, MaRTE permite manejar los dispositivos de red utilizando las funciones clásicas de manejo de ficheros (open/close/write/read/ioctl) (ver Figura 4.2). El acceso a esas funciones desde una aplicación Ada se realiza utilizando las funciones y procedimientos definidos en la adaptación a Ada del estándar POSIX, las denominadas *POSIX Ada bindings* (FLORIST 2011). El paquete `POSIX.IO` definido en los POSIX Ada bindings define las siguientes funciones y procedimientos:

- **Open:** Permite abrir una interfaz de red.

```
function Open
   (Name : POSIX.Pathname;
    Mode : File_Mode)
    return File_Descriptor;
```

Parámetros:

- `Name`: Nombre identificador de la interfaz de red en la forma `/dev/eth0`.
- `Mode`: Modo de apertura (lectura y/o escritura).
- `return`: Descriptor de fichero que se emplea para referirse a esta interfaz en otros procedimientos.

- **Close:** Cierra una interfaz de red referenciada por su descriptor de fichero.

```
procedure Close (File : in File_Descriptor);
```

- **Read:** Empleada para leer datos llegados a la interfaz de red especificada.

```
procedure Read
   (File   : in File_Descriptor;
    Buffer : out Ada.Streams.Stream_Element_Array;
    Last   : out Ada.Streams.Stream_Element_Offset);
```

Parámetros:

- `File`: Descriptor de fichero para la interfaz de red.
- `Buffer`: Array de bytes en el que se devuelven los datos leídos.
- `Last`: Número de bytes correctamente leídos.

- **Write:** Escribe datos en la interfaz de red y los envía.

```
procedure Write
   (File   : in File_Descriptor;
    Buffer : in Ada.Streams.Stream_Element_Array;
    Last   : out Ada.Streams.Stream_Element_Offset);
```

Parámetros:

- `File`: Descriptor de fichero para la interfaz de red.
- `Buffer`: Array de bytes que contiene los datos a escribir.
- `Last`: Número de bytes correctamente escritos.

- **Generic_Ioctl:** Procedimiento genérico definido por la llamada POSIX `Ioctl` (*input/ output control*). Esta llamada se emplea para permitir a una aplicación controlar o comunicarse con un driver mediante códigos y datos.

```
generic
   type Ioctl_Options_Type is (<>);
   type Ioctl_Data_Type is private;
   procedure Generic_Ioctl
      (File     : in     File_Descriptor;
       Request  : in     Ioctl_Options_Type;
       Data     : in out Ioctl_Data_Type);
```

Parámetros:

- `File`: Descriptor de fichero para la interfaz de red.
- `Request`: Número de código de requerimiento.
- `Data`: Valor o puntero que es pasado y retornado por el driver (parámetro in/out).

El kernel generalmente envía un `Ioctl` directamente al driver, el cual interpreta el número de requerimiento y los datos en función de como se haya especificado. El creador del driver documenta la función de cada número de requerimiento y los provee como constantes en un archivo de cabecera. En el caso de MaRTE, el desarrollador del driver dejó estas indicaciones en el archivo de cabecera `Ethernet_Driver.ads`.

Los requerimientos definidos en ese fichero permiten funciones bastante útiles, de las que en este proyecto se han empleado las siguientes:

- **Lecturas bloqueantes/No bloqueantes:** Permiten que las lecturas con el `Read` sean bloqueantes (el `Read` no retornará hasta que no haya recibido algún dato) y no bloqueantes (la llamada retornará haya o no haya datos).
- **Filtrado de protocolos:** Permite al driver filtrar los paquetes que se reciben en función del campo de protocolo definido en la trama Ethernet. En este proyecto, se filtró por paquetes IP, ya que son los únicos paquetes de la capa de red que se manejan. Esto aporta fiabilidad a la aplicación desarrollada, ya que debe preocuparse de realizar esta comprobación.

### 4.2.2. Modelado propuesto para las tramas

El núcleo de este proyecto lo conforman las tramas de datos. Apiladas constituyen la arquitectura de protocolos que permite las funcionalidades requeridas en el estándar GigE Vision como escrituras en registros, descubrimiento de dispositivos, transporte de imágenes, etc.

Fue necesario plantearse modelar la forma en que estas tramas debían ser representadas en la implementación Ada. Para ello, había que tener en cuenta los siguientes aspectos principales:

- Debían ser tramas cuyo **envío y recepción fuera sencillo**.
- Era necesario crear **funciones/procedimientos para obtener y/o establecer los valores de los campos** fácilmente para favorecer la manipulación y comprobación.
- El **esquema debía adaptarse** a las diferencias que por su propia definición existen entre las tramas. Algunas de ellas tienen tamaño fijo, pero otras cuentan con campos de tamaño variable o incluso campos cuya aparición es opcional. Esto obligó a cambiar el esquema adoptado en varias ocasiones para pasar de uno más rígido a otro más **flexible**.
- Las **tramas debían ser encapsulables**. Esto quiere decir que debían responder al modelo de protocolos apilados en que las tramas de nivel inferior encapsulan a las de niveles superiores.
- Se debían **minimizar las copias de los datos**. Este fue otro aspecto que acarreó varias modificaciones. Cuando de una trama se quiere extraer la trama que encapsula en su interior, se devuelve una copia de esos datos. Esto no es especialmente preocupante en el caso de tramas de control que ni son especialmente largas ni tienen requerimientos temporales estrictos. Sin embargo, en el caso de las tramas que transportan imagen resulta de vital importancia ya que la frecuencia a la que se pueden recibir imágenes está íntimamente relacionada con la eficiencia en este aspecto. Al hecho de realizar la copia, se une en este caso que los datos copiados tienen un tamaño considerable al tratarse de porciones de imagen. Por ello, para las tramas que transportan imagen se optó por evitar las copias y trabajar con punteros siempre que fuera posible.

Para cada trama se ha creado un paquete. De esta forma podemos reunir declaraciones de constantes, tipos y subprogramas referentes a un mismo protocolo en un único espacio. En Ada los paquetes cuentan con dos partes:

- **Parte visible o especificación:** Es la parte del paquete que visualiza el usuario. A su vez tiene una parte no visible o privada.
- **Cuerpo:** Es donde se implementan los detalles internos.

Esta forma de estructurar el código sigue la filosofía de la *programación modular*, en donde se diferencia entre una parte visible (la especificación del módulo) y una parte oculta que contiene los detalles de implementación.

A continuación se exponen las directrices que se han seguido para la definición de los campos y las tramas en el proyecto:

- **Definición de los campos de la trama**

  Aprovechando la posibilidad que Ada ofrece al usuario de definir **tipos de datos propios**, para cada campo de las tramas se ha definido un tipo de dato cuyas características de tamaño, estructura y valores permitidos respondieran a las definidas en los correspondientes estándares. De esta forma, se evitan problemas de diferencias entre lo indicado en el estándar y lo implementado.

  Así, si por ejemplo se toma el campo *puerto UDP de origen* de la trama UDP, se puede ver que el estándar define que su tamaño es de 2 bytes y que el rango de valores permitidos va desde 0 a 65535. Por ello se definió un tipo de dato que ocupara exactamente 2 bytes:

```
type UDP_Port is new MaRTE.Integer_Types.Unsigned_16;
```

  Algunos de los tipos de datos definidos pueden aprovecharse para ser utilizados con más de un campo, como ocurre con el propio puerto UDP. Por ello, en vez de definir tipos de datos distintos se especifica uno con un nombre más general que se usa en los campos puerto origen y destino.

  Hay tipos de datos que responden mejor a una **representación en forma de array de bytes**, como las direcciones IP que se componen de 4 valores de 1 byte. En estos casos, se define un array cuyo número de bytes totales sea el tamaño que ocupa el dato en cuestión.

```
type IP_Address is array (1 .. 4) of
   MaRTE.Integer_Types.Unsigned_8;
```

  Existe un problema que surge con los tipos de datos cuyo tamaño es superior a 1 byte y que es causado por la **distinta forma en que pueden ordenarse los bytes en memoria**. En las transmisiones a través de redes Ethernet, los protocolos asociados ordenan los bytes de la misma forma en que leemos el castellano: de izquierda a derecha (big-endian). Sin embargo, MaRTE al ejecutar sobre arquitectura PC ordena los bytes en sentido inverso (little-endian). Esto implica que al transcribir los datos de un lado a otro es **necesaria una reordenación** o de lo contrario la interpretación será incorrecta.

  Además de esto, la transmisión de datos ha sido pensada para **reutilizar memoria**. Por ello, cuando un dato va a ser enviado por la red, se emplea un puntero a la zona de

memoria en que está almacenado para que el driver utilice ese dato directamente sin copia de por medio. Esto obliga a tener en cuenta dos cosas:

- Los datos deben estar de forma **consecutiva en memoria**, sin espacios entre ellos.
- Los datos deben de tener el **orden big-endian** que la interfaz de red espera enviar.

Para solventar estos retos, se han distinguido dos casos en función del tamaño del campo:

- **Campos de tamaño múltiplo de 1 byte**
  - Campos que ocupan 1 byte: Pueden ser empleados en la transmisión sin modificación alguna.
  - Campos que ocupan más de 1 byte: Tienen dos definiciones. Por un lado, la definición pública del dato en la forma que se considere más oportuna y por otro una definición privada en formato de array de bytes (salvo que la pública ya fuera de array de bytes), que permiten un manejo interno más sencillo orientado a la reordenación. Con esta novedad la definición del campo UDP que se indicó anteriormente quedaría así:

    Parte pública:
    ```
    type UDP_Port is new MaRTE.Integer_Types.Unsigned_16;
    type UDP_Port_Array is private;
    ```

    Parte privada:
    ```
    type UDP_Port_Array is array (1 .. 2) of
    MaRTE.Integer_Types.Unsigned_8;
    ```

- **Campos de tamaño no múltiplo de 1 byte**

  En los protocolos empleados en este proyecto, estos campos vienen siempre consecutivos de forma que agrupados ocupan un tamaño múltiplo de 1 byte. De esta forma, pueden ser tratados como los datos indicados en el punto anterior. En la agrupación de los datos, se han encontrado dos situaciones:
  - Campos consecutivos que agrupados ocupan 1 byte: Es el caso de los campos IP *Version* (4 bits) y *Tamaño de cabecera* (4 bits).
  - Campos consecutivos que agrupados ocupan 2 o más bytes: Es el caso de los campos IP *Flags* (3 bits) y *Fragment Offset* (13 bits) de IP, que suman 2 bytes. Para proceder a mapearlos sobre un array de bytes, se agrupan como si fueran un solo campo de 1 byte. Por comodidad a la hora de programar, para asignar u obtener su valor, se usan procedimientos que asignan y obtienen sus valores a la vez. La definición de tipos quedaría así:

    Parte pública:
    ```
    type IP_Flags is range 0 .. 2 ** 3 - 1;
    type IP_Fragmnt_Offset is range 0 .. 2 ** 13 - 1;
    type IP_Flags_And_Fragmnt_Offset_Array is private;
    ```

    Parte privada:
    ```
    type IP_Flags_And_Fragmnt_Offset_Array is array (1 .. 2)
       of MaRTE.Integer_Types.Unsigned_8;
    ```

Como se puede observar, solo se han contemplado casos en que los datos (agrupados o no) ocupan bloques de datos múltiplos de 1 byte. Esto ocurre porque los protocolos de

comunicaciones definen los campos teniendo en cuenta esta consideración ya que resulta mucho menos engorrosa que acceder con desfases de bits. Un recurso que se suele emplear habitualmente es añadir datos de relleno a un campo hasta que el conjunto ocupe el tamaño esperado.

A pesar de todas estas consideraciones, el usuario no debe preocuparse de las representaciones internas de los datos que maneja, ya que son las funciones y procedimientos que los manipulan los que se encargan de garantizar el correcto orden y dejarlos listos para su almacenamiento y posterior lectura.

Por defecto en Ada no se asegura que los tipos de datos compuestos como `arrays` y `records` vayan a ocupar exactamente el tamaño equivalente a la suma de sus partes. Sin embargo, ya se ha indicado que es de vital importancia que todos los datos ocupen exactamente el tamaño mínimo sin espacios intercalados. Para este propósito Ada es un lenguaje muy apropiado, ya que uno de sus criterios de diseño fue que debía ser adecuado para programar sistemas empotrados, en los cuales el acceso al hardware es una parte fundamental de la programación.

Para obligar al compilador a adoptar esta representación, se han usado dos directivas proporcionadas por el lenguaje que garantizan que el espacio utilizado por un tipo es el deseado. Por un lado, el lenguaje permite indicar el espacio de memoria utilizado por un tipo de datos mediante la asignación de su atributo `Size`, lo obliga a que ese tipo ocupe el tamaño en bits especificado. Por otro lado, el `pragma` (directiva del compilador) de representación `Pack` garantiza que se emplea el espacio estrictamente necesario en vez de ubicar los datos según el criterio por defecto que da prioridad a lograr una mayor facilidad de acceso.

Ambas directivas son redundantes, pero así se asegura que el comportamiento es el esperado. Así, se tiene el siguiente ejemplo del tipo `UDP_Port_Array` compuesto por un array de dos bytes. Al ser un tipo compuesto, hay que explicitar el tamaño que se quiere utilizar.

```
type UDP_Port_Array is array (1 .. 2) of
   MaRTE.Integer_Types.Unsigned_8;
for UDP_Port_Array'Size use 16;
pragma pack (UDP_Port_Array);
```

- **Definición de la trama**

  En este proyecto se optó por implementar las tramas mediante un tipo de dato `record` de Ada. Los `record` son tipos compuestos que agrupan uno o más campos que pueden ser de cualquier tipo, incluidos tipo `array` o un nuevo `record`.

  En los `record` definidos, se recogen todos los campos con que cuenta la trama referidos mediante un nombre y su tipo de datos asociado. Como los `record` constituyen el espacio de memoria que almacena una trama y son pasados de forma directa a los procedimientos `Read` y `Write` del driver Ethernet, los campos deben definirse mediante los tipos de datos array siempre que ocupen más de 1 byte para que pueda facilitarse la reordenación y manejo. El siguiente es un ejemplo que muestra la definición de la trama IP:

```
type IP_Frame is record
   Version                  : IP_Version;
   Header_Length            : IP_Header_Len;
   TOS                      : IP_Type_Of_Service;
   Total_Length             : IP_Total_Length_Array;
   Id                       : IP_Identification_Array;
   Flags_And_Fragmnt_Offset :
      IP_Flags_And_Fragmnt_Offset_Array;
   TTL                      : IP_Time_To_Life;
   Proto_Type               : IP_Protocol;
   CRC                      : IP_Header_Checksum_Array;
   Source_Address           : IP_Address;
   Dest_Address             : IP_Address;
   Payload                 : IP_Payload (1 .. IP_Max_Payload);
end record;
```

Sin embargo, no se debe dar por sentado que los datos en el `record` se escribirán de forma secuencial y en el orden en que han sido especificados. El compilador de Ada puede decidir que, por ejemplo, por razones de eficiencia en el acceso a memoria se deban dejar espacios sin uso entre cada campo.

El lenguaje proporciona las llamadas “cláusulas de representación” que permiten forzar a nivel de bits la localización de los datos de un `record`. Por ejemplo, para el tipo `IP_Frame` mostrado con anterioridad se añade la siguiente cláusula de representación:

```
for IP_Frame use record
   Version at 0 range 4 .. 7;
   Header_Length at 0 range 0 .. 3;
   TOS at 1 range 0 .. 7;
   Total_Length at 2 range 0 .. 15;
   Id at 4 range 0 .. 15;
   Flags_And_Fragmnt_Offset at 6 range 0 .. 15;
   TTL at 8 range 0 .. 7;
   Proto_Type at 9 range 0 .. 7;
   CRC at 10 range 0 .. 15;
   Source_Address at 12 range 0 .. 31;
   Dest_Address at 16 range 0 .. 31;
   Payload at 20 range 0 .. (IP_Max_Payload * 8 - 1);
end record;
pragma pack (IP_Frame);
```

En ella se especifica para el `record` anteriormente definido el lugar que debe de ocupar cada campo siguiendo la siguiente construcción:

```
Nombre_Campo at Byte_Comienzo range Bit_Comienzo .. Bit_Fin
```

Donde:

- `Nombre_Campo` se refiere al campo de la trama a definir la posición.
- `Byte_Comienzo` es el índice del primer byte que ocupará el campo dentro de la trama completa.

- `Bit_Comienzo .. Bit_Fin` es el rango de bits definido para un campo teniendo como dirección base la de `Byte_Comienzo`. Se permite a un campo exceder espacio del byte base si es de un tamaño mayor:
  ```
  Source_Address at 12 range 0 .. 31;
  ```

A resaltar también el caso en que dos campos comparten el mismo byte, como ocurre con *Version* y *Header_Length*. La posición de cada uno es definida a nivel de bits:

```
Version at 0 range 4 .. 7;
Header_Length at 0 range 0 .. 3;
```

Al igual que en la definición de los campos de tipo array, en este caso se emplea `pragma pack (IP_Frame)` para agrupar los datos de forma que ocupen su mínima extensión posible. No debería ser necesario emplearlo tras haber especificado el dato que se asocia a cada byte del `record`, pero se ha puesto para asegurarse más si cabe ante comportamientos no esperados.

Por último, citar la existencia del campo *Payload*. Algunas tramas como Ethernet, IP o UDP encapsulan tras su cabecera los datos de otro protocolo de nivel superior cuyo tamaño es indefinido, aunque con un límite máximo. Otras tramas, aunque no encapsulen protocolos si que cuentan con campos cuyo tamaño es variable. Ejemplo de ello es el campo *Opciones* de la trama DHCP.

Para estos casos se han definido campos de tamaño variable, que en el caso de Ethernet, IP y UDP se ha denominado *Payload* (carga útil) y que se definen como un tipo array de bytes:

```
type IP_Payload is new Ada.Streams.Stream_Element_Array;
```

En el `record`, se han definido como un array del tamaño máximo permitido:

```
Payload : IP_Payload (1 .. IP_Max_Payload);
```

Como se ha comentado anteriormente, Ada permite estructurar las declaraciones en parte pública y privada, así como ocultar las implementaciones. El siguiente es el esquema que se ha seguido para definir los procedimientos y funciones, tipos de datos, constantes, etc.:

- **Especificación** (declaraciones):
  - **Parte pública** (accesible por el usuario):
    - Definición de los tipos de los campos de las tramas. Es el tipo de dato que el usuario utilizará cuando quiera obtener o modificar el valor que tiene un campo. Ej:
      ```
      type IP_Total_Length is new
        MaRTE.Integer_Types.Unsigned_16;
      ```

- Funciones y procedimientos para obtener (`Get`) o modificar (`Set`) el valor de los campos de las tramas. Ej:

```
function Get_Total_Length (Frame : in IP_Frame)
                           return IP_Total_Length;

procedure Set_Total_Length (Total_Length : in
                           IP_Total_Length;
                           Frame : in out IP_Frame);
```

- Otras funciones y procedimientos públicos como los que calculan el CRC, chequean el mensaje o lo muestran por pantalla.
- Constantes útiles como el tamaño máximo de cabecera, puertos UDP por defecto, etc. Ej:

```
Eth_Max_Length : constant
   Ada.Streams.Stream_Element_Offset := 1514;
```

- **Parte privada** (no accesible por el usuario):
  - Definiciones de los tipos privados de los campos de las tramas. Ej:

```
type IP_Identification_Array is array (1 .. 2) of
   MaRTE.Integer_Types.Unsigned_8;
for IP_Identification_Array'Size use 16;
pragma pack (IP_Identification_Array);
```

  - Tipo `record` definiendo la trama. Ej:

```
type UDP_Frame is record
   Source_Port      : UDP_Port_Array;
   Destination_Port : UDP_Port_Array;
   Length           : UDP_Length_Array;
   Checksum         : UDP_Checksum_Array;
   Payload         : UDP_Payload (1 .. UDP_Max_Payload);
end record;
for UDP_Frame use record
   Source_Port at 0 range 0 .. 15;
   Destination_Port at 2 range 0 .. 15;
   Length at 4 range 0 .. 15;
   Checksum at 6 range 0 .. 15;
   Payload at 8 range 0 .. (UDP_Max_Payload * 8 - 1);
end record;
pragma pack (UDP_Frame);
```

  - Constantes privadas como direcciones de registros o valores por defecto de algunos campos que el usuario no necesita conocer.

- **Cuerpo** (implementaciones): Implementaciones de los procedimientos y funciones cuyas cabeceras se definieron en la especificación. Nótese en el siguiente ejemplo para fijar el valor del campo *Total Length*, que al ocupar este 2 bytes tiene que haber reordenación para almacenarlo en la trama.

```
procedure Set_Total_Length (Total_Length : in
                               IP_Total_Length;
                            Frame : in out IP_Frame) is
   Array_Origin : IP_Total_Length_Array;
   for Array_Origin'Address use Total_Length'Address;
   Array_Reorder : IP_Total_Length_Array;
begin
   Array_Reorder (1) := Array_Origin (2);
   Array_Reorder (2) := Array_Origin (1);
   Frame.Total_Length:=Array_Reorder;
end Set_Total_Length;
```

### 4.2.3. Minimización de copias de datos

Hasta este momento se han visto las posibilidades del driver Ethernet para obtener o escribir datos de o en la interfaz de red así como las reglas generales que han guiado la definición de las tramas en el proyecto. Queda por ver la forma en que se conectan los datos que manejan unas capas u otras.

Se propone un ejemplo concreto de intercambio de datos entre capas, pues su posible solución será extrapolable al resto de intercambios. En el ejemplo de la Figura 4.3 aparece el espacio de memoria donde el driver Ethernet deja los datos leídos (`Unformat_Received_Frame`) y el espacio reservado para la trama Ethernet (`Eth_Frm`).

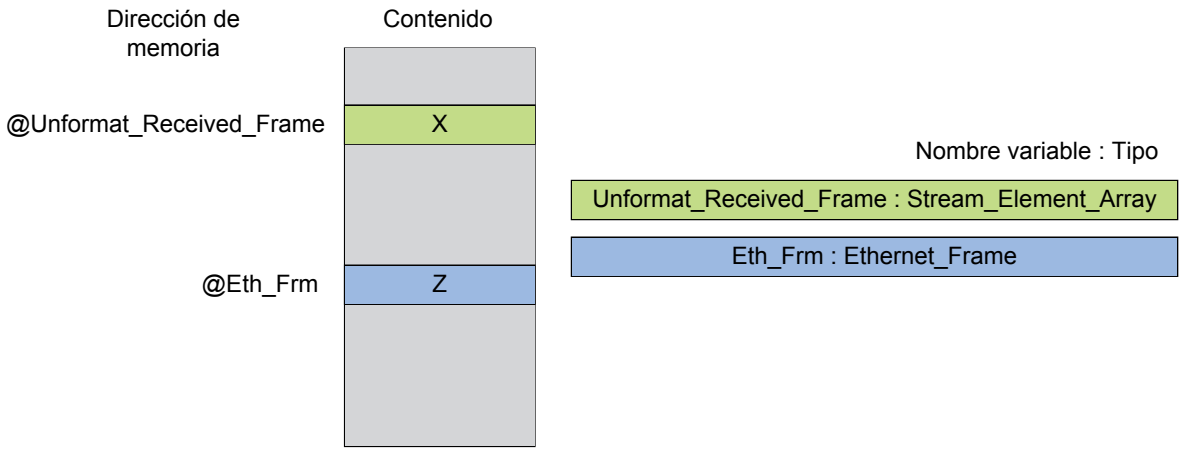


**Figura 4.3: Espacios de memoria reservados para el array de bytes leídos de la interfaz y para la trama Ethernet.**

La primera idea que puede surgir es una asignación:

```
Eth_Frm := Unformat_Received_Frame;
```

pero son tipos de datos distintos cuya composición impide en Ada que con una conversión de tipo sea suficiente.

En este punto, puede surgir la idea de realizar esa conversión de tipo de forma manual. Esta consistiría en recorrer el array de bytes e ir extrayendo los bytes uno a uno e introduciéndolos en los campos del `record` Ethernet.

Sin embargo, Ada permite hacerlo de una forma mucho más sencilla mediante asignación de atributos `Address`. El atributo `Address` aplicado sobre cualquier objeto, procedimiento, función, paquete o cualquier otra entidad permite retornar la dirección del primer elemento que compone ese objeto. Además, puede ser empleado para modificar la dirección de ese objeto.

Un ejemplo de esta útil sentencia se encuentra en el procedimiento `Read` de Ethernet, en el que se han definido las siguientes variables:

```
Eth_Frm : out UDP_Over_Eth.Eth_Frame.Eth_Frame;
Unformat_Received_Frame : Ada.Streams.Stream_Element_Array
   (1 .. UDP_Over_Eth.Eth_Frame.Eth_Max_Length);
```

Mediante la asignación del atributo `Address` se establece la dirección de la trama Ethernet para la dirección del `Unformat_Received_Frame` que ha sido inicializado llamando al `Read` del driver Ethernet. De esta forma, se obtienen dos perspectivas de la misma zona de memoria, una como `Ethernet_Frame` y otra como `Stream_Element_Array` (Figura 4.4):

```
for Unformat_Received_Frame'Address use Eth_Frm'Address;
```

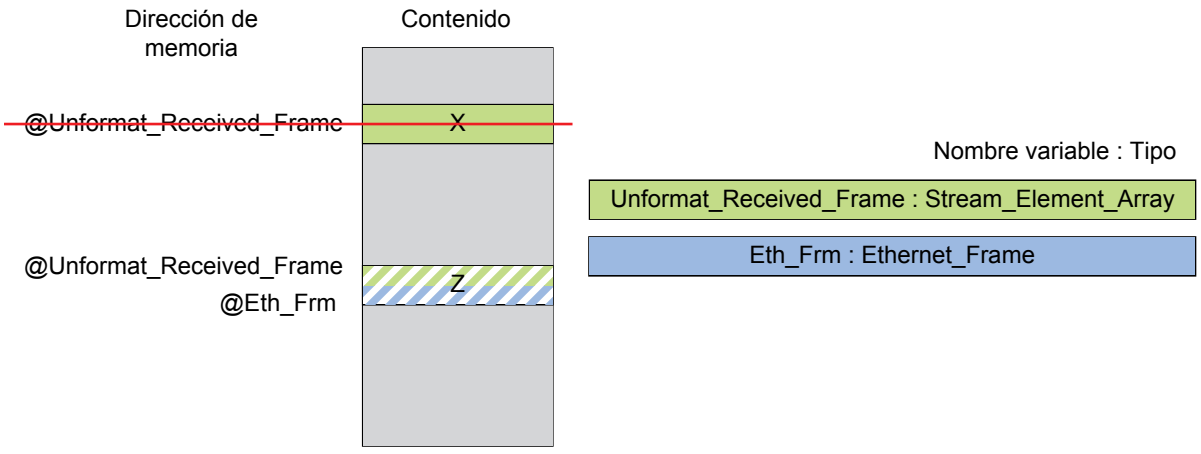


**Figura 4.4: Mediante el uso de punteros se evita la copia de los datos de gran tamaño que manejan estas tramas.**

Empleando este esquema se pueden referenciar unas capas con otras sin necesidad de realizar en este proceso copias adicionales de los datos.

Otro ejemplo se tiene con la obtención de la trama IP a partir de la carga útil de Ethernet. Primero hay que inicializar dicha carga útil mediante la función `Get_Payload` de la trama Ethernet:

```
Eth_Payload : UDP_Over_Eth.Eth_Frame.Eth_Payload (1 ..
   UDP_Over_Eth.Eth_Frame.Eth_Max_Payload);
Eth_Payload := UDP_Over_Eth.Eth_Frame.Get_Payload (Frame =>
   Eth_Frm);
```

Con esta forma de proceder se puede ya acceder a la trama IP:

```
IP_Frm : UDP_Over_Eth.IP_Frame.IP_Frame;
for IP_Frm'Address use Eth_Payload'Address;
```

El mismo proceso puede repetirse de forma inversa cuando lo que se quiere no es obtener los datos encapsulados sino encapsularlos. En ese caso, con sucesivas llamadas a los procedimientos `Set` de las tramas se inicializan los campos que después con `for-use` sobre los atributos `Address` se van atribuyendo a las capas inferiores.

Llegados a este punto, la referenciación entre capas está solventada y podemos hacer uso de la pila de protocolos completa.

Sin embargo, existe un problema con este sistema relacionado con las copias de datos y que se hizo relevante cuando se procedió a la captura de imágenes. La gran cantidad de paquetes transmitidos en este proceso implica que los tiempos de procesamiento de los datos recibidos deben de ser muy pequeños para no perjudicar el rendimiento. Las funciones o procedimientos que obtienen los campos de las tramas (`Get`) devuelven siempre copias de los datos (Figura 4.5).

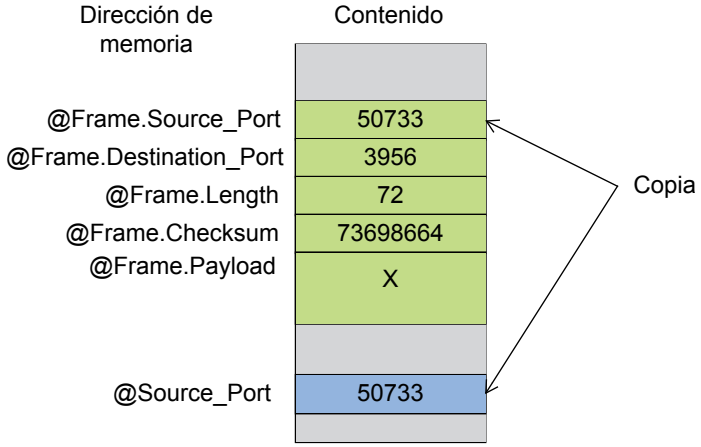


**Figura 4.5: Tras la ejecución del `Get_UDP_Source_Port`, se obtiene una copia del campo del Frame UDP `Source_Port`.**

De esta forma, el dato se encuentra duplicado en dos zonas de memoria con la consiguiente pérdida de tiempo en realizar el duplicado y poca eficiencia en la gestión de la memoria al emplear el doble de espacio.

Que estos hechos se produzcan con variables como el puerto UDP no tiene importancia, pues es un dato de un tamaño muy reducido. Sin embargo, ejecutar un `Get_Payload` en una trama implica la copia de datos de mucho mayor tamaño, ya que se está copiando todo el contenido que encapsula ese protocolo.

Visto este problema, se planteó la solución teniendo en cuenta los siguientes pasos:

- Analizando cuál era el número mínimo de copias que no se podían evitar.
- Empleando punteros para referirse a aquellas copias que no se podían evitar.

El análisis condujo a la situación que se muestra en la Figura 4.6.

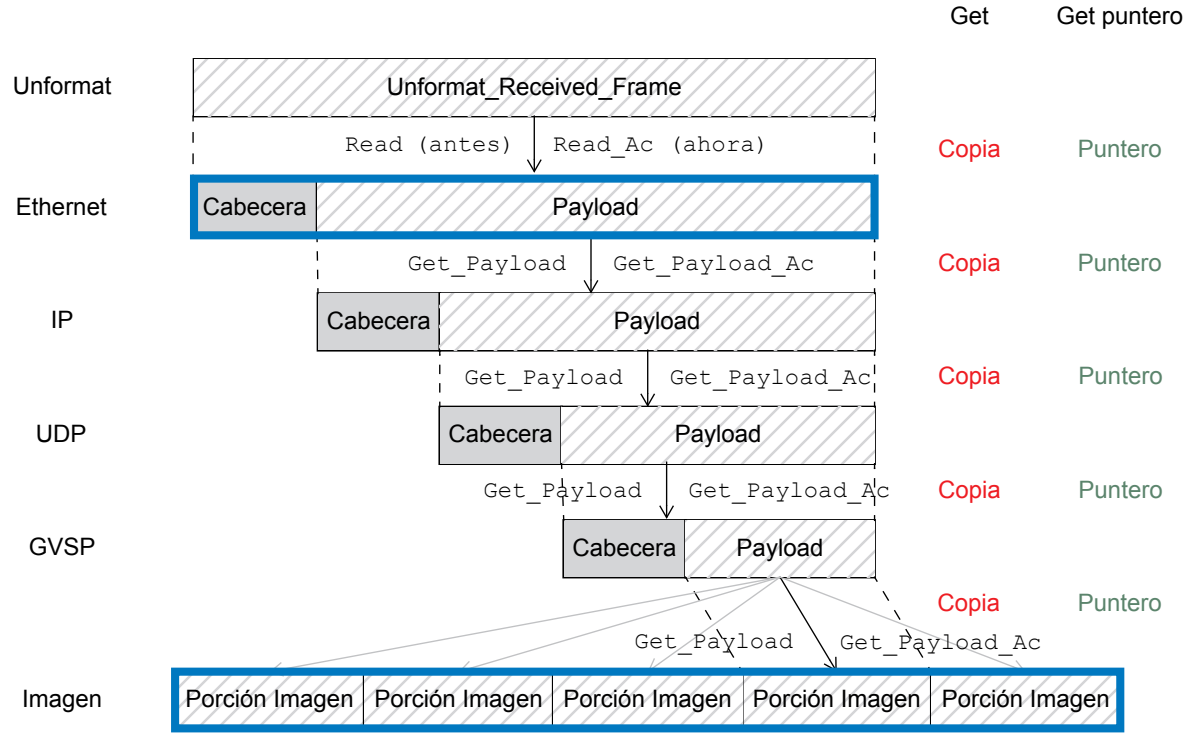


**Figura 4.6: Esquema que representa las zonas de memoria definidas para la imagen y las tramas manejadas en el proceso de captura. Muestra, para las operaciones necesarias si se realizaba una copia para devolver el resultado antes y si se realiza ahora. Se resaltan en azul las zonas de memoria cuya definición es obligatoria.**

Dos son las zonas de memoria que como se puede ver en la Figura 4.6 en azul es obligatorio definir actualmente. Los procedimientos `Get_Payload/Read` antes devolvían una copia cada vez que retornaban sus datos. Actualmente, `Get_Payload_Ac/Read_Ac` se limitan a referenciar mediante punteros las zonas resaltadas en azul.

La explicación de por qué estas dos zonas de memoria deben existir es la siguiente. Por un lado en el caso del driver la copia es necesaria debido a su diseño interno: no es posible retornar un puntero al paquete ya que una vez leído puede ser reutilizado su espacio en el buffer por si llegara uno nuevo. Por otro lado en el caso de la aplicación se debe reservar una zona en la que almacenar la imagen completa sobre la cual se copian de forma consecutiva todas las porciones recibidas en los distintos paquetes.

### 4.2.4. Herramienta de análisis de protocolos Wireshark

En un proyecto en que el medio para comunicar datos entre la aplicación y el dispositivo es la red Ethernet y los paquetes que ambos intercambian son la base que permite recibir imágenes o ejecutar comandos, el uso de un analizador de protocolos de red se convierte en una gran ayuda para el desarrollo. La función básica de los analizadores de protocolos es capturar los paquetes que viajan por la red y mostrar su contenido de la forma más detallada posible.

*Wireshark* (WIRESHARK 2011) es un analizador de protocolos libre. Permite capturar en tiempo real el tráfico que llega a una interfaz de red seleccionada, identificando los protocolos de las tramas recibidas y analizando y mostrando sus contenidos. Incluye un lenguaje para filtrar y ordenar los protocolos que se reciben. Además, permite guardar las capturas realizadas para poder visualizarlas en cualquier otro momento.

Wireshark ha sido empleado en el PC de desarrollo sobretodo en las primeras fases del proyecto en que aún no se utilizaba la cámara y se estaba programando la pila UDP/IP. En aquel momento, el objetivo era modelar y construir una pila de protocolos que permitiera a MaRTE enviar y recibir tramas.

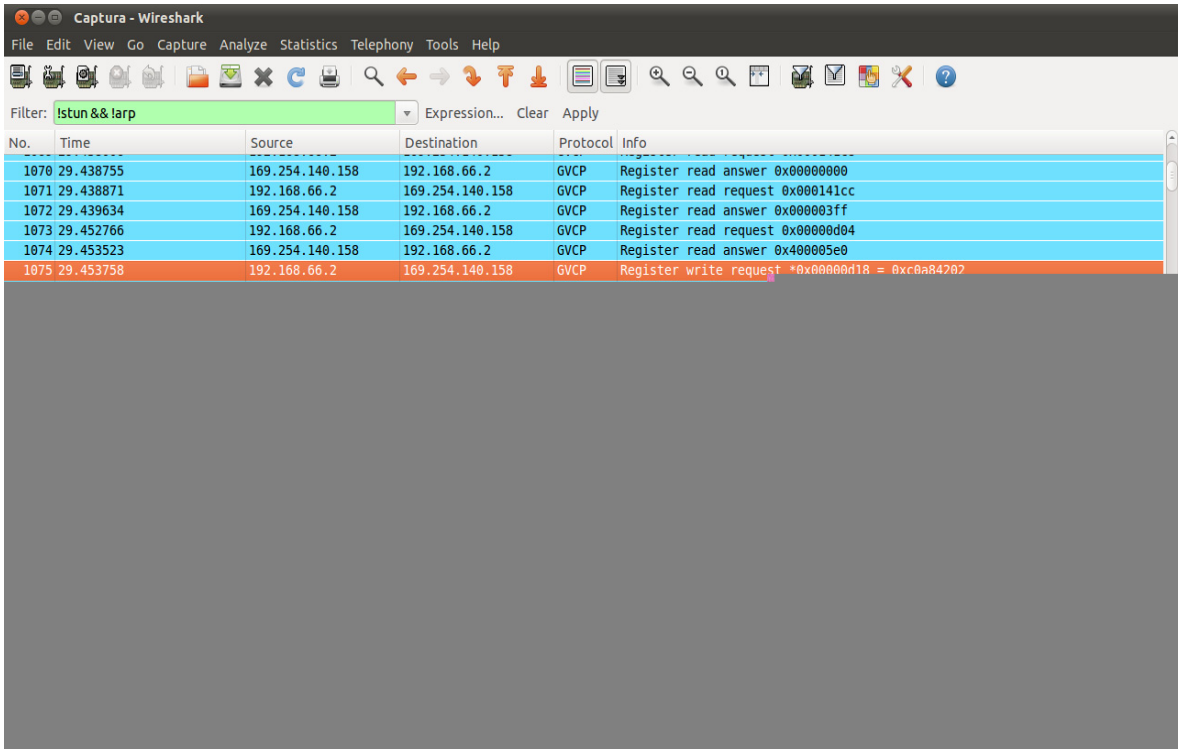


**Figura 4.7: Captura de los paquetes intercambiados entre la cámara y la aplicación mediante Wireshark.**

Para el caso de envío de tramas desde MaRTE, Wireshark permitía comprobar si las tramas construidas eran correctas o no, por lo que constituyó una herramienta fundamental de depuración.

En el sentido contrario (para recibir tramas en MaRTE) se empleó otra herramienta de software libre llamada *packETH* (PACKETH 2011) que desde el PC de desarrollo permitía enviar

paquetes totalmente configurables de diversos protocolos. Después en MaRTE podían descomponerse para comprobar si se estaban recibiendo correctamente.

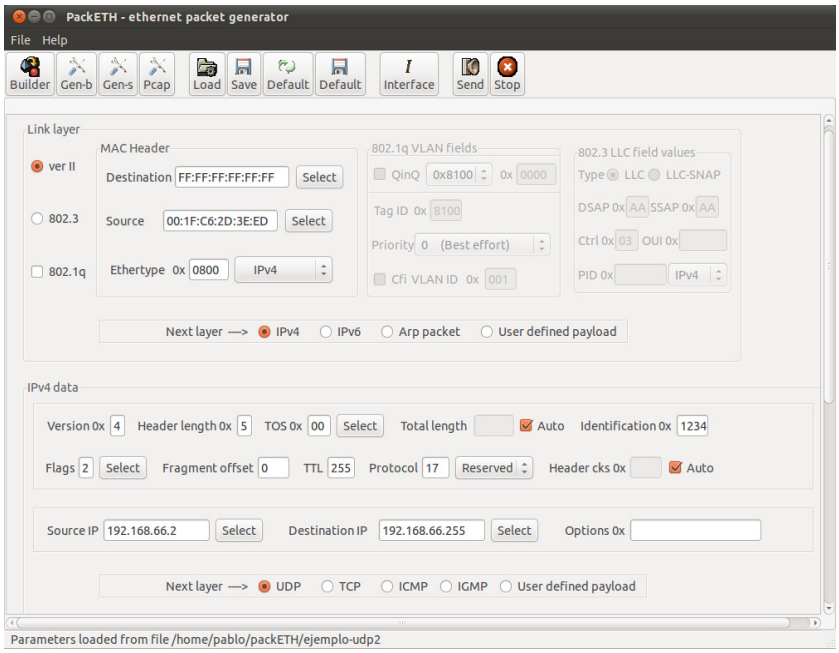


**Figura 4.8: Generación de un paquete UDP de prueba con packETH.**

Los protocolos que el Wireshark reconoce se van ampliando. En una primera versión del programa manejada, los protocolos de GigE Vision GVCP y GVSP no eran detectados. Sin embargo, las últimas versiones de prueba ya los reconocían. A pesar de ello, es probable que al tratarse de un estándar no abierto, no les haya sido permitida una implementación con alto nivel de detalle en el reconocimiento de los campos de las tramas.

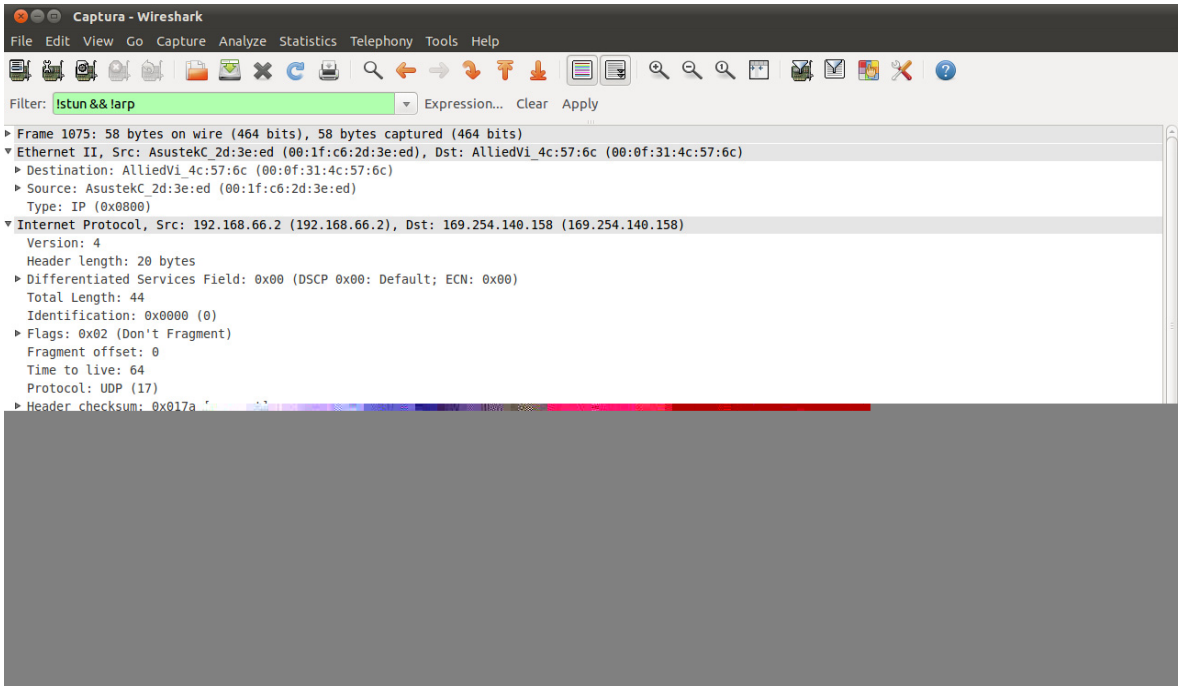


**Figura 4.9: Detalles de un paquete GVCP capturado con Wireshark.**

## 4.3. Implementación de los protocolos

Los protocolos que es necesario implementar en el proyecto se han estructurado en paquetes. La Figura 4.10 permite visualizar los paquetes creados y la jerarquía que existe entre ellos. Los paquetes situados en la parte superior utilizan los de la parte inferior. En todo momento se ha pensado en realizar módulos que pudieran ser reutilizables y ampliables sin dificultad.

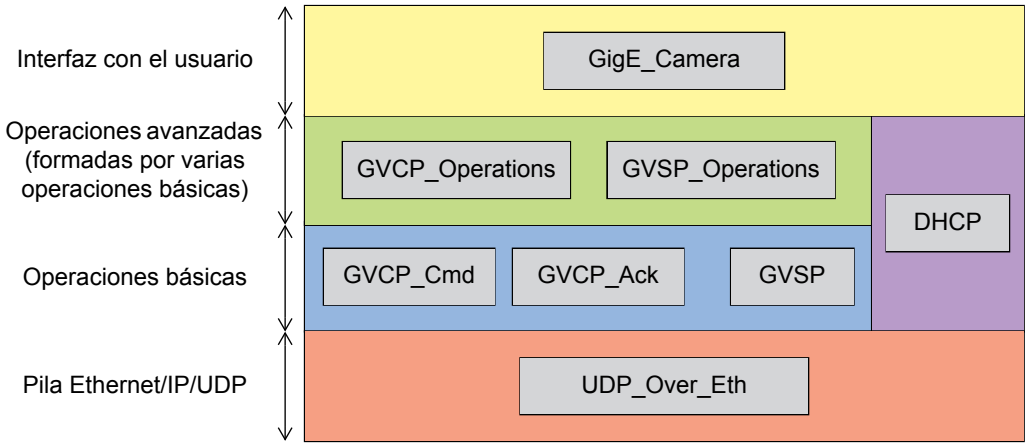


**Figura 4.10: Jerarquía de los paquetes implementados.**

En la base, la pila Ethernet, IP y UDP constituye un módulo software independiente que se puede emplear en otros proyectos. Lo mismo ocurre con DHCP.

En cuanto a GigE Vision, se proporciona una interfaz cómoda para el usuario que permite realizar las operaciones más habituales. Esta interfaz se apoya en los paquetes de operaciones avanzadas que a su vez se apoyan en paquetes con operaciones más básicas. De esta forma, si se quisiera añadir una nueva operación, el proceso consistiría en crear un procedimiento en `GigE_Camera`, quizá alguna operación en los paquetes `Operations` y después basarse en la infraestructura que proporcionan los paquetes de operaciones básicas (acceso a registros, etc.).

### 4.3.1. Ethernet, IP y UDP

La Figura 4.11 muestra la estructura de paquetes que compone la pila de protocolos Ethernet, IP y UDP. Existe un paquete de nivel superior (`UDP_Over_Eth`) que solo recoge las definiciones de direcciones MAC e IP, pues son empleadas por varios de los paquetes hijo.

Los paquetes `Ethernet`, `IP` y `UDP` tienen como tipos más relevantes los correspondientes`record` (`Ethernet_Frame`, `IP_Frame`, `UDP_Frame`) que reúnen todos los campos como tipo privado. Por ello, son necesarias funciones y procedimientos para manejar estos datos. `IP` y `UDP` cuentan además con sendas funciones (`To_X_Frame_Ac`) para convertir el puntero que señala al campo de datos del protocolo que les encapsula (Ej: campo de datos de Ethernet, que encapsula a IP) a puntero al `record` correspondiente al frame (Ej: frame IP). Ambos también cuentan con funciones para calcular el CRC. Por su parte, `UDP` cuenta con `Is_UDP_Frame`. Como comentamos anteriormente, el driver Ethernet permite filtrar los datos recibidos por protocolo, con lo que ya hemos asegurado que los paquetes recibidos son IP. A partir de ese punto, se deben comprobar sin dar por sentado los protocolos esperados. Con esta función aplicada a una trama, comprobamos si el frame contiene al protocolo UDP.

Por último, el paquete `Operations` es el que lleva a cabo el desarrollo de las posibilidades que el driver Ethernet ofrecía. `Ioctl_Protocol` e `Ioclt_Block` son instancias del procedimiento genérico `Ioclt` que permiten que la interfaz filtre por protocolo y sea bloqueante respectivamente.

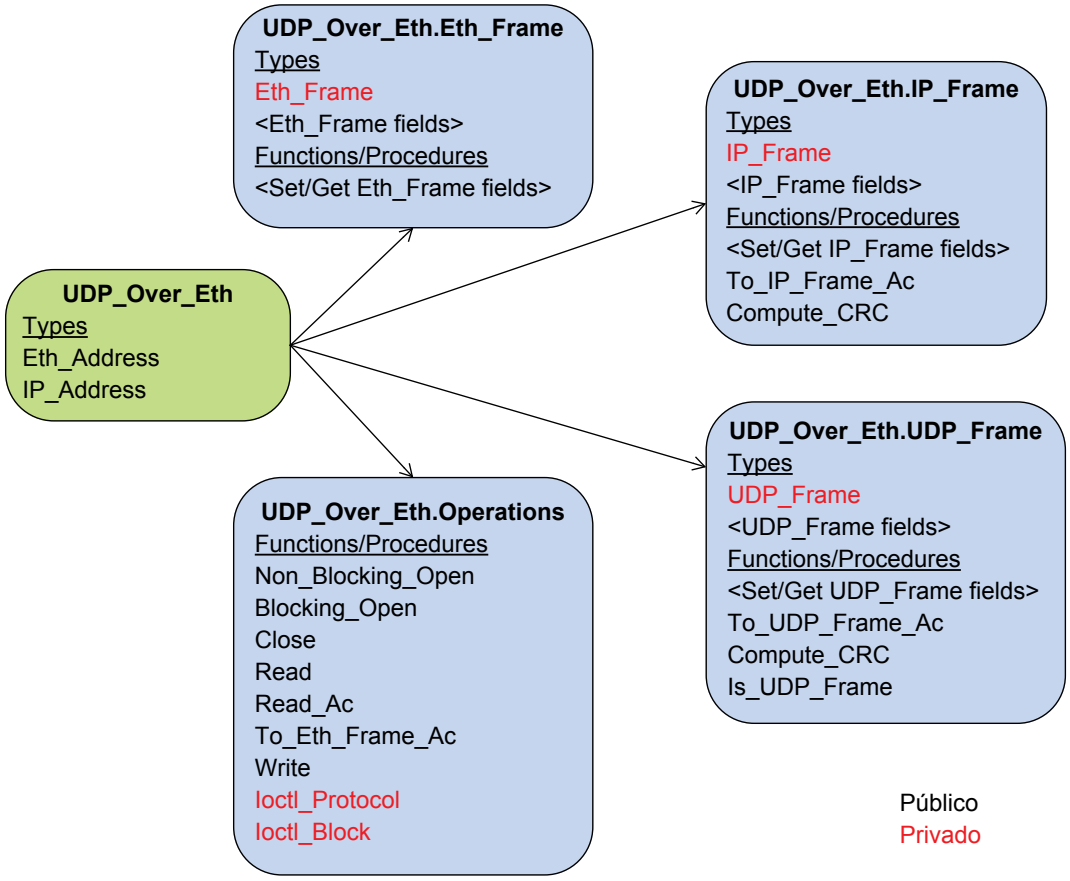


**Figura 4.11: Esquema paquetes de la pila Ethernet, IP y UDP.**

### 4.3.2. DHCP

En la Figura 4.12 se puede ver el esquema del único paquete creado para otorgar a la cámara su direcionamiento utilizando el protocolo DHCP. Este paquete permite obtener y modificar todos los campos de una trama DHCP. Entre los más interesantes se encuentran las opciones que extienden la información que el mensaje puede transportar. De acuerdo al estándar GigE Vision, un dispositivo debe poder procesar obligatoriamente las opciones 1 (subnet mask) y 3 (router), por lo que se han implementado. Haciendo uso del Wireshark, se pudo comprobar que en el intercambio realizado entre la cámara y el driver Linux que el fabricante permite descargar estaban presentes otras opciones, por lo que aunque con estas dos ya es suficiente se decidió implementarlas. Estas son todas las que se explicaron en el Apartado 2.3.5, "DHCP". Las opciones se representan mediante el `record DHCP_Option`. Con procedimientos `Create_Option` se crean `records` para cada tipo de opción. Con los procedimientos `Get_Option` se busca la opción en cuestión en el mensaje recibido y si se encuentra se retorna el valor leído.

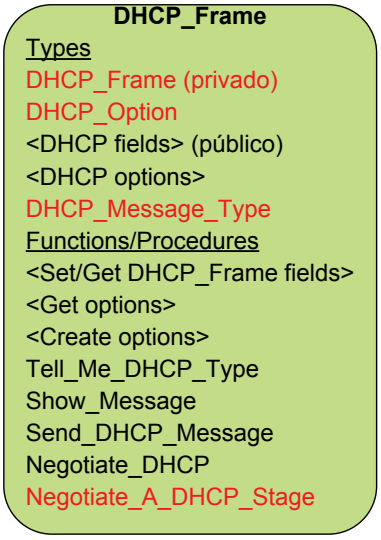


**Figura 4.12: Paquete DHCP_Frame.**

El procedimiento `Tell_Me_DHCP_Type` distingue si el mensaje recibido es de tipo Discover o Request. Además, realiza algunas comprobaciones como chequeo de CRC o longitud del mensaje. Con `Show_Message` se pueden mostrar los campos que componen un mensaje DHCP.

El procedimiento `Send_DHCP_Message` toma como parámetro el mensaje recibido desde la cámara y envía la respuesta correspondiente. Las opciones DHCP enviadas en este mensaje son siempre las mismas, adaptadas a los requerimientos de la cámara concreta que ha sido manejada, por lo que no aporta la inteligencia suficiente como para contestar mensajes de DHCP con requerimientos más complejos que le pudieran llegar de otros dispositivos. El procedimiento `Negotiate_DHCP` realiza el proceso completo de intercambio de los 4 mensajes DHCP. Este último procedimiento es el único que necesita utilizar el usuario de la cámara, los demás se proporcionan porque son utilizados por el propio procedimiento y por si el usuario desea realizar la negociación de forma manual.

Aunque el paquete está dirigido a responder las necesidades de direccionamiento de la cámara, podría ser utilizado para otras aplicaciones que requieran DHCP en el caso de que no precisen parámetros más allá de los básicos (IP, máscara de subred y router). En caso contrario, es fácilmente extensible y no debe suponer problema alguno negociar otro tipo de parámetros.

## 4.3.3. GVCP

### 4.3.3.1. GVCP Command

Los paquetes que conforman los comandos GVCP se muestran en la Figura 4.13. La cabecera común a todos los campos se recoge en el paquete `GVCP_Cmd`. El resto de paquetes correspondientes a sus respectivos comandos definen el resto de campos propios de cada uno y funciones para modificarlos. En estos paquetes se han añadido procedimientos que permiten enviar fácilmente los mensajes más habituales. Estos procedimientos se encargan de crear e inicializar todos los campos de la pila de protocolos instanciada en cada caso y enviar el mensaje. De esta forma, se le ahorra al usuario tener que realizar este trabajo de forma manual. Solo se le pide que especifique siempre ciertos parámetros como direcciones IP o puertos UDP y, dependiendo del tipo de trama enviado, algunos campos concretos como por ejemplo la dirección del registro a consultar. El siguiente es un ejemplo de cabecera para uno de estos procedimientos, en concreto el que envía un mensaje que permite leer un registro:

```
procedure Send_Read_Register_Message (
   Reg_Addr           : in GVCP_Cmd_Memory_Address;
   Network_Fd         : in POSIX_IO.File_Descriptor;
   GVCP_Req_Id        : in GVCP_Cmd_Req_Id;
   Client_MAC         : in UDP_Over_Eth.Eth_Address;
   Server_MAC         : in UDP_Over_Eth.Eth_Address;
   Client_IP          : in UDP_Over_Eth.IP_Address;
   Server_IP          : in UDP_Over_Eth.IP_Address;
   Server_Control_Port: in UDP_Over_Eth.UDP_Frame.UDP_Port);
```

Como se comentó en el resumen del estándar del Apartado 3, “Estándar GigE Vision”, a partir de una determinada dirección de memoria del dispositivo, los registros definidos dependen del fabricante. GigE Vision define los registros para manejar los canales de control, streaming y mensajes, direccionamiento del dispositivo, velocidad de transmisión, etc. Sin embargo, cuando se procedió a implementar un mecanismo de capturas, se vio que no estaban especificados registros para indicar el formato de pixel, dimensiones de la imagen o incluso para dar la orden de comenzar la captura. Con el Wireshark y empleando los drivers que el fabricante proporciona para Linux, se constató la presencia de comandos que escribían en registros sin identificar justo antes de comenzar la captura. Por ello, se contactó con el fabricante de la cámara (responsable de definir estos registros) que cedió la especificación de los registros. Esto permitió manejar la funcionalidad completa de la cámara.

De las tramas de comando de que dispone GVCP, se han implementado todas salvo `Packet_Resend`, `Pending`, `Action` y `Readmem`, pues no se consideró necesario contar con ellas para el objetivo propuesto. El soporte para cada una de las tramas se proporciona en el paquete correspondiente:

- **`Discovery_Frame:`** Dispone de un procedimiento que envía el mensaje de descubrimiento.
- **`Forceip_Frame`**: Forceip ha sido implementado para aprovechar una de las características más interesantes que este mensaje ofrece, la de permitir el reinicio del proceso por el cual la cámara obtiene una dirección IP. Para ello, en `Send_Restart_IP_Configuration_Message` se envía un mensaje en el que el campo `StaticIP` es `0.0.0.0`, según indica el estándar. A diferencia de otros mensajes,

en que se ha implementado el reconocimiento, en Forceip no se ha hecho por lo que debe ser indicado en el mensaje enviado para que la cámara no responda.

- **`Readreg_Frame`:** Permite enviar mensajes para leer uno o varios registros (la cámara empleada lo soporta) cuyas direcciones se especifican en los parámetros. La implementación permite a su vez permite leer el registro CCP, algo bastante común pues refleja el nivel de control que la aplicación tiene sobre el dispositivo.
- **`Writereg_Frame`:** Permite enviar comandos de escritura de registros. Es un comando de vital importancia y eso se refleja en el tipo de mensajes que se pueden enviar. Apertura/cierre de canales de control y streaming o reseteo de la cámara. Cuenta con un comando que escribe en el registro que da orden de iniciar la captura.
- **`Readmem_Frame`:** Cuenta con un procedimiento que envía un mensaje que lee tantos bytes como se especifiquen a partir de una determinada dirección de memoria.

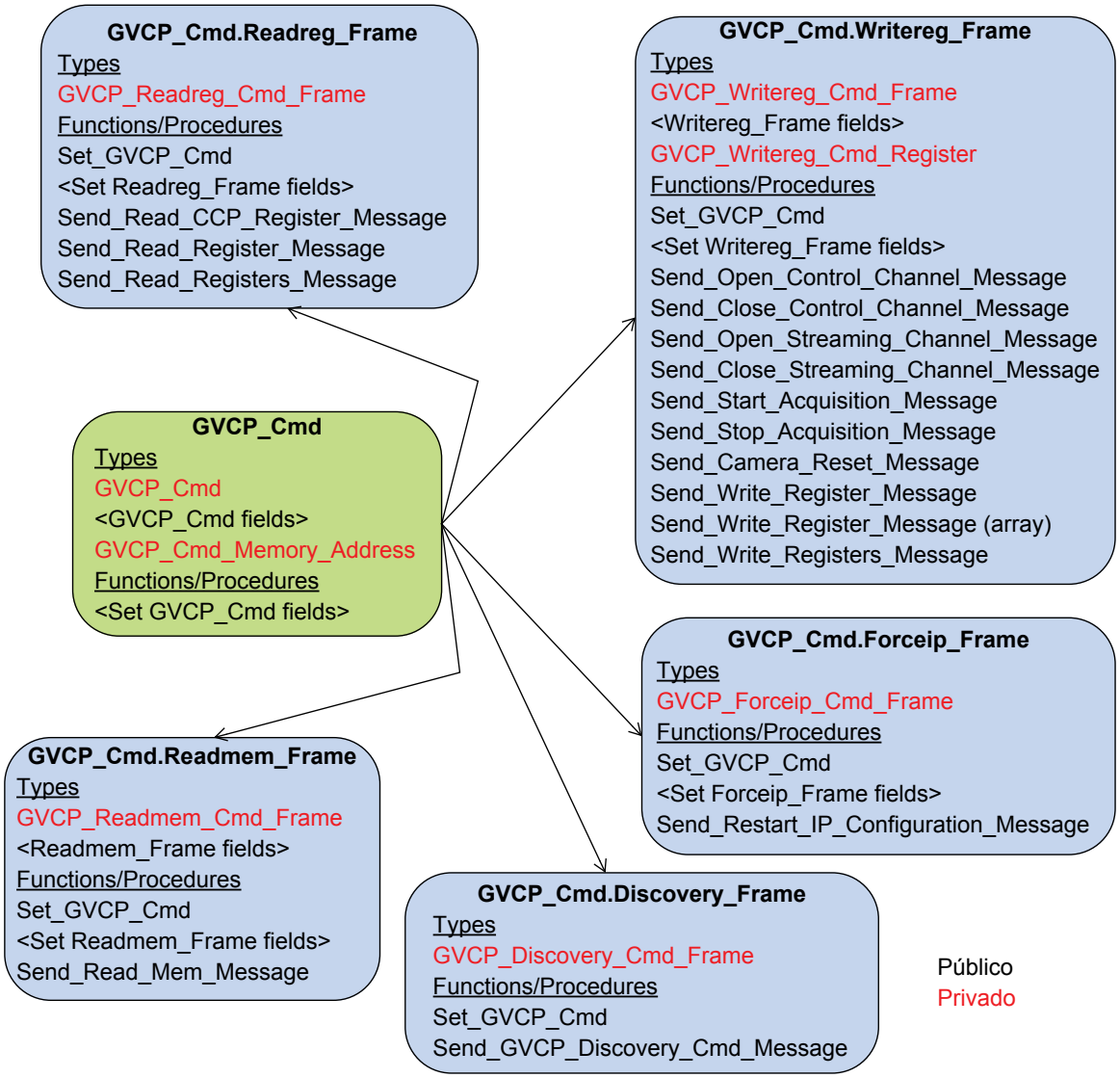


**Figura 4.13: Esquema paquetes de GVCP Command.**

### 4.3.3.2. GVCP Acknowledge

Los paquetes que representan las respuestas de reconocimiento que envía la cámara se muestran en la Figura 4.14. Al igual que en los mensajes de comando, los reconocimientos cuentan con un paquete padre que recoge la parte de la trama común a todos ellos. Además, este paquete define un tipo de dato enumerado `GVCP_Ack_Message_Type` que toma los valores `Discovery`, `Readreg`, `Readmem`, `Writereg` y `No_GVCP_Ack`.

El paquete cuenta con la función `Tell_Me_GVCP_Ack_Type`, cuya labor consiste en:

1. Comprobar utilizando el `Is_UDP_Frame` del paquete `UDP_Frame` que el mensaje recibido es UDP.
2. Verificar el CRC de IP y el de UDP si tiene.
3. Comprueba que el puerto de origen es el puerto por defecto que usan los dispositivos GVCP para el canal de control. Comprueba que el puerto destino es el mismo que especificamos al abrir el canal de control. Si las comprobaciones han sido satisfactorias, podemos asegurar que el paquete ha sido enviado a la aplicación que se ha desarrollado y no a otra presente en el PC.
4. Consulta el campo `Acknowledge` para averiguar el tipo de mensaje recibido de entre los especificados en el enumerado.

En caso negativo en cualquiera de los puntos, se retorna un `No_GVCP_Ack`.

Los paquetes hijo, aparte de los campos y operaciones para obtener los datos recibidos, cuentan con procedimientos para mostrar y comprobar el mensaje recibido. `Show_Message` muestra los campos del mensaje por pantalla, algo que es bastante útil para labores de depuración. Por su parte, los `Check_Message` realizan un segundo chequeo sobre el contenido del mensaje tras el realizado por `Tell_Me_GVCP_Ack_Type` consistente en:

1. Comprueba si el `Ack_Id` es menor que el esperado. En este caso, el mensaje se descarta (no se clasifica como erróneo) ya que se considera que si se está esperando un mensaje posterior es porque ya se ha procesado el recibido. Esta situación es perfectamente viable cuando enviamos en más de una ocasión un mismo comando (mismo `Req_Id`) porque no estamos recibiendo el reconocimiento correspondiente. En ese caso podemos recibir (aunque sea con retraso) un reconocimiento por cada uno de ellos.
2. Comprueba si el `Ack_Id` es el esperado.
3. Comprueba el código de estado que el dispositivo retorna. Si es 0, no ha habido error.
4. Comprueba que la longitud del mensaje recibido es acorde a lo esperado.
5. En función del mensaje:
   - `Writereg_Frame` comprueba el número de registros correctamente escritos.
   - `Readmem_Frame` comprueba que la dirección base de lectura es la esperada.

Se retornan valores booleanos que indican si el mensaje ha sido descartado y si ha sido comprobado con éxito.

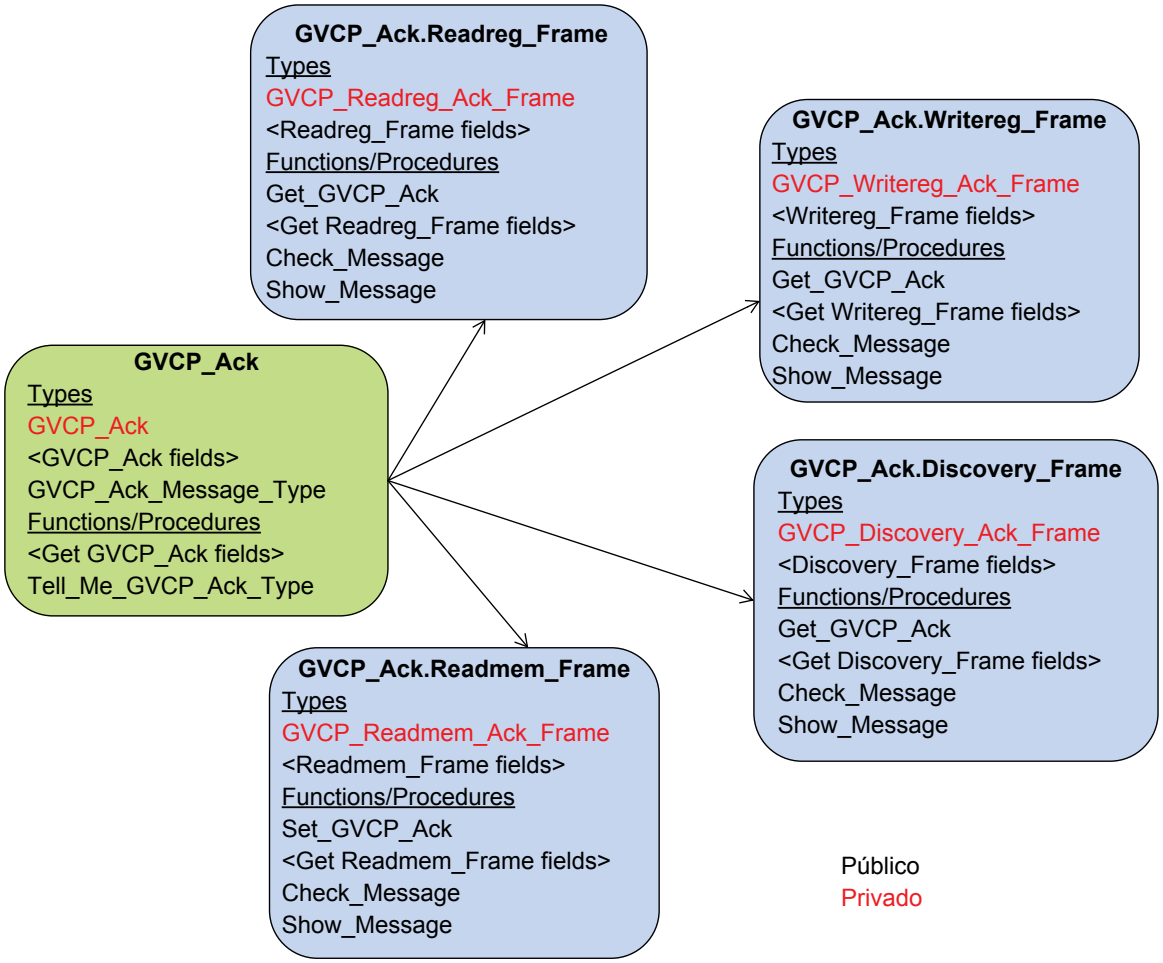


**Figura 4.14: Esquema paquetes de GVCP Acknowledge.**

### 4.3.4. GVSP

La Figura 4.15 muestra el árbol de paquetes para implementar las tramas de GVSP. El paquete GVSP es la cabecera y padre del resto de paquetes. A sus campos se adosan los de los `Image`, `Payload` y `Trailer` correspondientes. Como se pudo ver en el Apartado 3.4.1, "Bloques de datos", existen distintos tipos de datos que una cámara puede enviar. De acuerdo con los objetivos de este proyecto, se ha optado por el tipo de dato Image, pues transporta imágenes en crudo de acuerdo a formatos conocidos como RGB24 o Mono8, los dos que han sido aceptados en el proyecto sin perjuicio de que otros se puedan emplear en un futuro con sencillos cambios. Por tanto, las únicas tramas GVSP implementadas corresponden al tipo Image.

La trama de `Payload_Image` es hija directa de GVSP ya que no define ningún campo adicional en función del tipo de dato transportado. Por su parte, las tramas `Image_Frame` y `Trailer_Frame` heredan de `Leader` y `Trailer` respectivamente pues estos últimos definen campos adicionales.

Las funciones `Tell_Me_X_Type` de cada paquete distinguen si el mensaje recibido contiene alguna de las tramas de los paquetes hijo y en ese caso a cuál de ellas. Al igual que con los paquetes `GVCP_Ack`, se realizan comprobaciones sobre si es UDP, los CRC y los puertos.

A su vez, los paquetes finales también muestran el contenido recibido mediante `Show_Message`. Los `Check_Message` realizan las siguientes comprobaciones:

1. Se comprueba el código de estado que devuelve el dispositivo. 0 significa que no hubo problemas.
2. Esta fase de la comprobación depende del tipo de mensaje:
   - **`GVSP.Leader.Image_Frame`:** Se comprueba que el formato de pixel y las dimensiones de la imagen son las esperadas.
   - **`GVSP.Payload_Image_Frame`:** Se comprueba la porción recibida pertenece a la imagen que se comenzó a recibir con el paquete `Leader`.
   - **`GVSP.Trailer.Image_Frame`:** Se comprueba que esta terminación corresponde a la imagen que se especificó en `Leader`.

En caso de que alguna de las comprobaciones no sea satisfactoria, se retorna un valor booleano `False`.

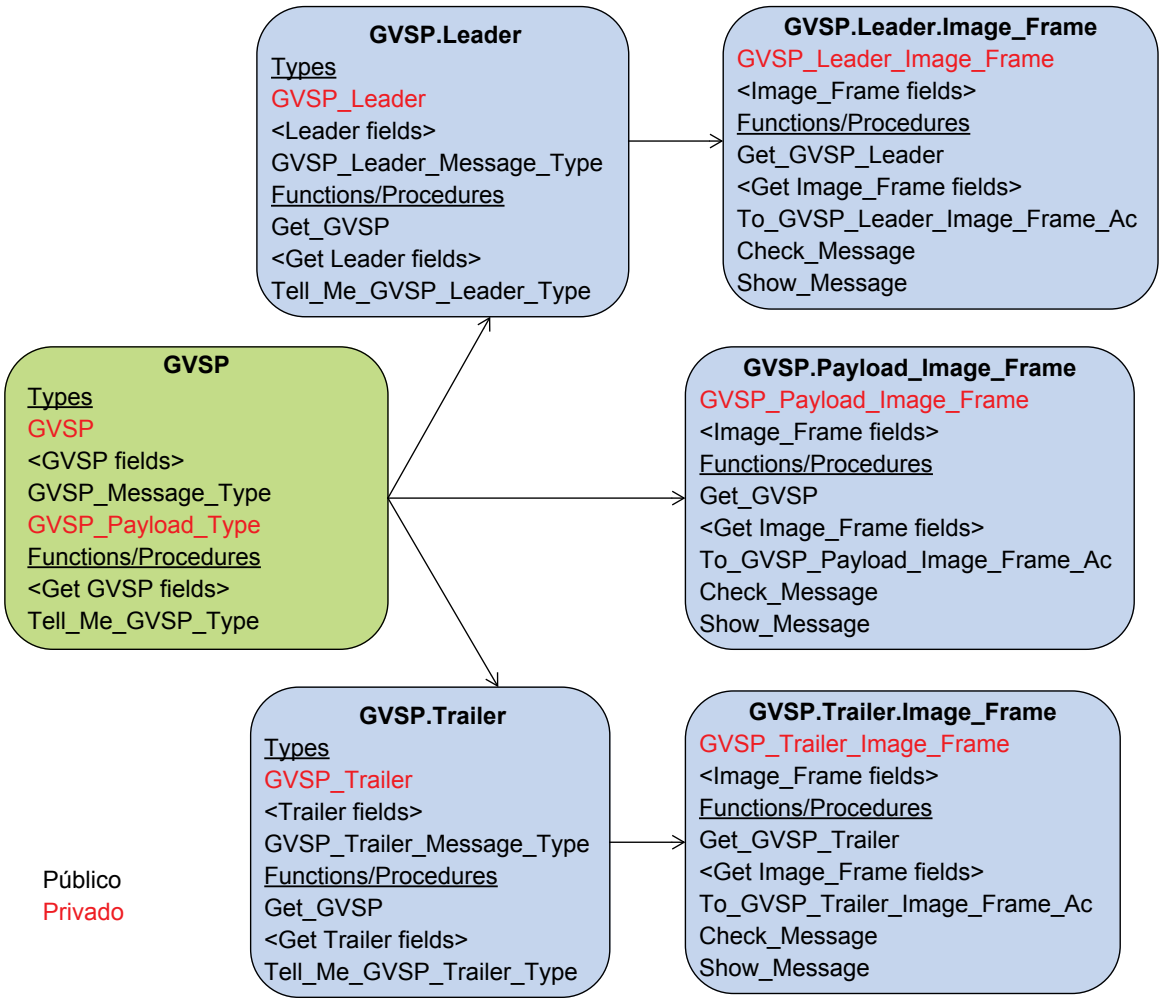


**Figura 4.15: Esquema paquetes de GVSP.**

# 4.4. Implementación de paquetes que hacen uso de GVCP y GVSP

Con la estructura de paquetes hasta ahora definida, ya es posible hacer total uso de las características de GigE Vision planteadas en este proyecto. Sin embargo, para darle más funcionalidad se han definido nuevos paquetes que proporcionan operaciones de más alto nivel utilizando las operaciones básicas definidas por los paquetes GVCP y GVSP de los apartados anteriores. Estas operaciones, más complejas, permiten desde capturar una imagen completa a escribir en un registro utilizando espera y comprobación de respuestas.

## 4.4.1. Paquete GVCP_Operations

Permite ejecutar comandos GVCP con comprobación de resultados, es decir, lleva a cabo el envío del comando y espera la recepción del reconocimiento.

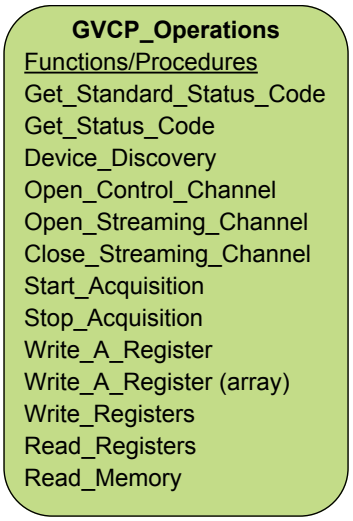


**Figura 4.16: Paquete GVCP_Operations.**

De acuerdo a las recomendaciones del estándar, se debe esperar al mensaje de reconocimiento un máximo de 200 ms. Pasado ese tiempo, se puede reenviar el mensaje de nuevo hasta un máximo de 3 envíos. En caso de que este proceso no sea satisfactorio, se debe avisar al usuario. En cada una de las operaciones definidas en este paquete se ha seguido este mismo esquema. El pseudocódigo de las operaciones es el siguiente:

```
para i=1 hasta i=3 hacer
   enviar_comando;
   mientras siempre hacer
      lectura_no_bloqueante;
      si datos_leidos>0 entonces
         si tipo_mensaje=tipo_mensaje_esperado entonces
            comprobar_mensaje;
            si no mensaje_descartado entonces
               mensaje_esperado + 1;
               retornar;
            fsi
         fsi
      sino
         salir_mientras;
      fsi
   fmientras
   esperar_200_ms;
   lectura_no_bloqueante;
   si datos_leidos>0 entonces
      si tipo_mensaje=tipo_mensaje_esperado entonces
         comprobar_mensaje;
         si no mensaje_descartado entonces
            mensaje_esperado + 1;
            retornar;
         fsi
```

```
      fsi
   sino
      salir_mientras;
   fsi
fpara
```

La operación de comienzo de adquisición, además de escribir en el registro que hace que la cámara comience a enviar imágenes, escribe previamente todos los parámetros que se ha permitido modificar en la imagen: formato de pixel, altura y anchura de la imagen, posiciones X e Y de la esquina superior derecha de la imagen, tamaño del paquete de datos y velocidad de envío de datos en bytes por segundo.
El `Standard_Status_Code` que devuelven los mensajes de reconocimiento que envía la cámara es almacenado sobre una variable. También existe otra variable denominada `Status_Code` que permite a las operaciones dejar códigos de error si lo desean. Por ejemplo, la operación de comienzo de adquisición que se acaba de citar escribe esta variable para indicar en cuál de las fases que debe realizar ha fallado (los códigos específicos se encuentran en el código fuente, concretamente en las cabeceras).
Estas dos variables se pueden consultar con las funciones `Get_Standard_Status_Code` y `Get_Status_Code`.

## 4.4.2. Paquete GVSP_Operations

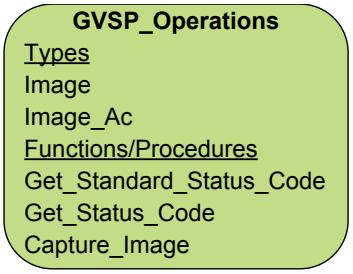


**Figura 4.17: Paquete GVSP_Operations.**

Cuenta con un procedimiento que captura una imagen. De los paquetes que recibe, busca un `Leader` a partir del cual comienza a capturar la imagen hasta que se encuentre el `Trailer` de la misma imagen. Las porciones de imagen que se reciben entre ambos paquetes se almacenan de forma posicional en el espacio designado. Si hay pérdidas de algunas porciones de la imagen no sucede nada. Además, dado que se utiliza siempre el mismo espacio de memoria para almacenar la imagen, las porciones no recibidas suelen estar ya inicializadas con porciones recibidas anteriormente (que normalmente tienen un contenido similar). En el caso de que un paquete `Leader` llegue de nuevo sin que se haya recibido el `Trailer` del anterior, se comienza a adquirir una nueva imagen.

Esta lógica se resume en el siguiente pseudocódigo:

```
mientras siempre hacer
   lectura_bloqueante;
   si datos_leidos>0 entonces
      si
         mensaje_leido=cabeza ->
            si mensaje_leido=cabeza_imagen entonces
               si comprobar_mensaje_leido=correcto entonces
                  imagen_actual=imagen_mensaje_leido;
               si no
                  retornar fallo;
               fsi
            fsi
```

```
            mensaje_leido=datos ->
               si mensaje_leido=datos_imagen entonces
                  si recibi_cabeza_imagen entonces
                     si comprobar_mensaje_leido=correcto entonces
                        extraer_porcion_imagen;
                     si no
                        retornar fallo;
                     fsi
                  fsi
               fsi

            mensaje_leido=final ->
               si mensaje_leido=fin_imagen entonces
                  si recibi_cabeza_imagen entonces
                     si comprobar_mensaje_leido=correcto entonces
                        retornar imagen;
                     si no
                        retornar fallo;
                     fsi
                  fsi
               fsi
         fsi
      fsi
   fmientras
```

Al igual que con las operaciones de GVCP, se dejan a disposición del usuario dos códigos de error que puede consultar con las funciones `Get_Standard_Status_Code` y `Get_Status_Code`.

### 4.4.3. Paquete Gige_Camera

`Gige_Camera` constituye el paquete de mayor nivel en la jerarquía y su finalidad es la de servir como interfaz con el usuario. Fue creado pensando en la forma en que un usuario esperaría utilizar un driver como el desarrollado. Permite enmascarar bajo procedimientos de uso más simple las capacidades que los paquetes `GVCP_Operations` y `GVSP_Operations` ofrecen.

Además, redefine los tipos de datos que el usuario utiliza para que no tenga que manejar otros que se encuentran repartidos por todos los paquetes del driver y con tipos más bajo nivel de abstracción. Para el formato de imagen, define un `record Image_Format_Params` con los campos formato de pixel, dimensiones y origen superior izquierdo de la imagen. Algunos `records` de este tipo han sido inicializados con formatos por defecto. Se han definido por ejemplo `MONO8_320x240` y `RGB8PACKED_640x480`.

Al igual que con los paquetes `GVCP_Operations` y `GVSP_Operations`, se pueden consultar los códigos de error devueltos por la cámara o por los procedimientos en la última operación realizada. Así, en el caso de que cualquiera de las operaciones falle, se pueden consultar estas variables para más información.

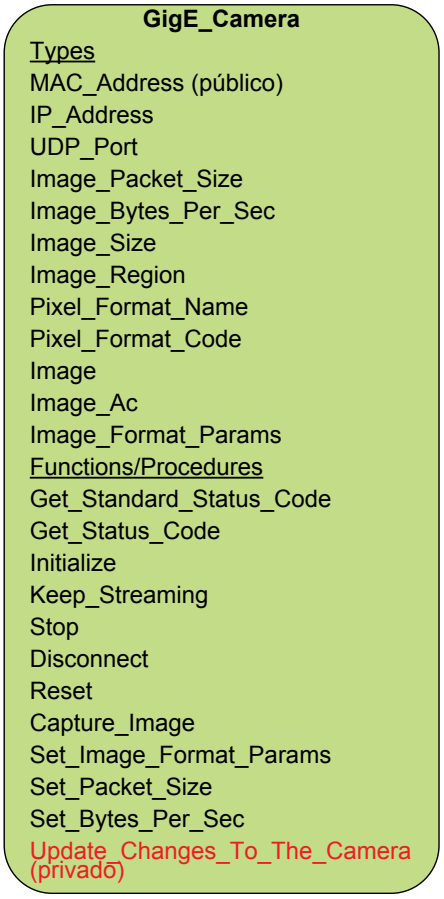


**Figura 4.18: Paquete Gige_Camera.**

A continuación se realiza una breve explicación de las operaciones definidas en el paquete:

- **Initialize:** El usuario le pasa los parámetros que se van a utilizar durante todo el programa como son direcciones MAC e IP o los puertos del PC a emplear. Emplea el `record` predefinido `MONO8_320x240` como formato por defecto inicial. Después, realiza el proceso siguiente:

```
si descubrimiento_dispositivo=fallo entonces
   enviar_mensaje_reiniciar_configuracion_ip;
   si negociar_dhcp=exito entonces
      si descubrimiento_dispositivo=fallo entonces
         retorna fallo;
      fsi
   si no
      retorna fallo;
   fsi
fsi
si abrir_canal_control=fallo entonces
   retorna fallo;
fsi
si abrir_canal_streaming=fallo entonces
   retorna fallo;
fsi
```

```
si comenzar_adquisicion=fallo entonces
   retorna fallo;
fsi
retorna exito;
```

- **Keep_Streaming:** Como se comentó al describir el estándar (Apartado 3.3.1.1, "Canal de control"), es necesario mantener una comunicación cada cierto tiempo (en la cámara manejada en el proyecto 3.5 s) en el sentido PC a cámara. Este procedimiento envía un mensaje de lectura de un registro. El usuario no debe olvidarse de enviarlo o la cámara dejará de capturar imágenes.
- **Stop:** Detiene la adquisición de imágenes en la cámara.
- **Disconnect:** Detiene la adquisición de imágenes en la cámara y cierra los canales de streaming y control.
- **Reset:** Envía un mensaje que escribe en el registro que reinicia la cámara a sus ajustes por defecto.
- **Capture_Image:** Toma una captura de imagen.
- **Update_Changes_To_The_Camera:** Realiza una llamada al procedimiento que comienza las adquisiciones que, se recuerda, también establece los parámetros de formato de imagen, velocidad de captura y tamaño del paquete de imagen. Previamente, realiza una llamada a `Stop` para detener la adquisición en curso si se estaba realizando alguna. Este procedimiento privado es el empleado por los procedimientos que modifican parámetros de captura: `Set_Image_Format_Params`, `Set_Packet_Size` y `Set_Bytes_Per_Sec`.

# 4.5. Patrones de uso

Este apartado pretende mostrar unos breves pseudocódigos sobre cómo debería el usuario emplear (desde el más alto nivel) los procedimientos cuya implementación se ha explicado.

Para utilizar la cámara, hay que comenzar siempre con una inicialización. Esta se realiza con unos parámetros por defecto. Si no se está de acuerdo con ellos, se pueden modificar (`Set_Image_Format_Parameters`), al igual que se modifican también la velocidad de transmisión (`Set_Bytes_Per_Second`) y el tamaño del paquete transmitido (`Set_Packet_Size`). Estas modificaciones se pueden realizar en cualquier momento: se hayan capturado anteriormente imágenes o no.

Lo habitual es emplear el procedimiento de captura de forma continuada para obtener una secuencia de video, por lo que se suele localizar en un bucle. Es extremadamente importante enviar el recordatorio `Keep_Streaming` periódicamente de forma que pasen menos de 3.5 segundos (en esta cámara) entre cada mensaje si no queremos perder el contacto con la cámara.

Se exponen dos ejemplos comunes de utilización:

- **Patrón de uso básico:** Consiste en capturar una imagen según los parámetros que por defecto utiliza el driver.

```
Gige_Camera.Initialize;
mientras siempre hacer
   Gige_Camera.Capture_Image;
   hacer_trabajo_que_se_desee_con_la_imagen
   si hay_que_mantener_contacto entonces
      Gige_Camera.Keep_Streaming;
   fsi
fmientras
```

- **Patrón para cambiar cualquier parámetro de captura**, por ejemplo la región de interés, tras varias capturas previas.

```
Gige_Camera.Initialize;
mientras siempre hacer
   Gige_Camera.Capture_Image;
   hacer_trabajo_que_se_desee_con_la_imagen
   si hay_que_cambiar_region_de_interes entonces
      Gige_Camera.Set_Image_Format_Params;
   fsi
   si hay_que_mantener_contacto entonces
      Gige_Camera.Keep_Streaming;
   fsi
fmientras
```

# 5. Evaluación y pruebas

Las pruebas realizadas durante todo el proceso de desarrollo se llevaron a cabo empleando una cámara GigE Vision modelo Manta G-032 de Allied Vision Technologies. De acuerdo a su especificación, permite una tasa de transmisión de imágenes de 80 fps a su máxima resolución que es de 0.3 megapíxel (656x492) con una relación de aspecto de 4:3 en formato RGBa de 32 bits con conexión Ethernet de 1 Gbps.


**Figura 5.1: Cámara Manta G-032 empleada en el proyecto.**

No se han manejado características fuera de lo definido en el estándar, pues corresponden en la mayor parte de los casos a ajustes finos de la imagen (como ganancias o balances de colores) específicos de este modelo de cámara en cuestión. El usuario, sin embargo, conociendo los registros que almacenan estas características podría modificarlos utilizando las operaciones descritas en el capítulo anterior.

Las pruebas realizadas consisten en hacer uso de los procedimientos y funciones que constituyen la interfaz con el usuario del software desarrollado, las cuales se han definido en el paquete `Gige_Camera` (Apartado 4.4.3, “Paquete Gige_Camera”). El modo en que han sido empleadas es similar al descrito en el Apartado 4.5, “Patrones de uso”.

Para comprobar de visualmente que las capturas realizadas eran correctas se creó un paquete que permitía mostrar la imagen. Este paquete emplea el driver VGA (*Video Graphics Adapter*) de MaRTE para mostrar imágenes en los dos formatos manejados en el proyecto `MONO8` y `RGB8PACKED`. Con `MONO8`, solo es necesario indicarle al driver VGA que emplee el modo blanco y negro para mostrar correctamente cada píxel. Sin embargo, con el formato a color, hubo que hacer una conversión de RGB a formato VGA. Dado que VGA dispone de varias paletas de color, se empleó la de 16 colores para facilitar la conversión de un formato a otro y porque ya cubría el propósito de comprobar que la captura era correcta.

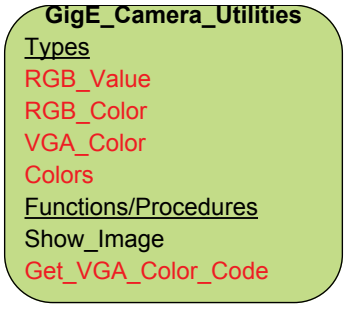


**Figura 5.2: Paquete GigE_Camera_Utilities.**

Tres son las pruebas que se ha dejado a disposición del usuario del driver para que examine su funcionamiento. Su cometido es el siguiente:

1. **Prueba mostrando la imagen:** El programa consta de dos tareas, la tarea principal realiza capturas continuas de imágenes en formato RGB24 y otra tarea adicional muestra los resultados de la captura (Figura 5.3). La prioridad de la tarea adicional es más baja que la de la tarea que captura para no entorpecer este proceso y permitir únicamente mostrar la imagen cuando la captura no necesite los recursos. Además el programa modifica el formato de captura tras un tiempo para comprobar que la imagen cambia de formato.

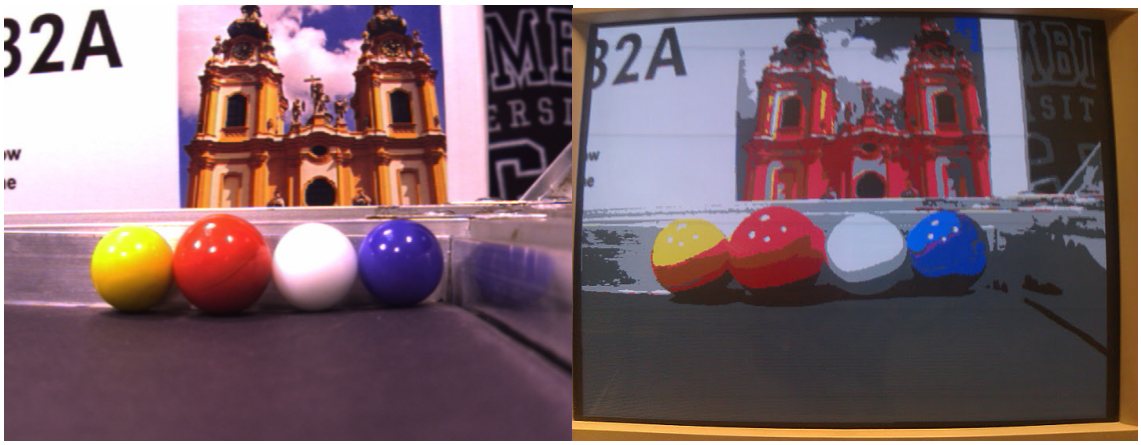

**Figura 5.3: Imagen capturada mediante el driver de Linux (izquierda) y mediante el driver desarrollado para MaRTE (derecha). Ambas han sido capturadas con la misma calidad, pero en MaRTE se ha representado utilizando VGA de 16 colores.**

2. **Prueba continua para detectar errores:** Muestra estadísticas cada cierto intervalo de capturas. Permite visualizar el número de capturas realizadas y de ellas, las exitosas y los errores. Modifica formatos de captura y velocidades sobre la marcha.
3. **Prueba de medición de tiempos:** Realiza un banco de pruebas sobre distintos formatos de imagen y resoluciones, midiendo los tiempos de captura y de otros procesos. En el siguiente apartado se muestran y comentan los resultados obtenidos en este test.

# 5.1. Medida de prestaciones

En sistemas de tiempo real es fundamental conocer los tiempos de ejecución de las operaciones a realizar para poder realizar el análisis que permita concluir si un sistema es planificable o no.

Se han realizado medidas de tiempos de las operaciones más importantes. En todos los casos se han medido los *tiempos de ejecución* y *totales*. La diferencia entre estos dos tipos de medidas estriba en que la primera contabiliza el tiempo de ejecución que el programa ha necesitado para realizar una operación, mientras que el segundo mide el tiempo que ha pasado desde que comenzó la operación hasta que la terminó. Este último no implica que todo el tiempo la aplicación se esté ejecutando sino que también se contabiliza la espera por bloqueos que, en el caso de la aplicación desarrollada, se producen esperando a que lleguen paquetes de la imagen.

La medidas se han realizado en un computador con procesador Pentium III a 800MHz y 256 MB de RAM.

En la Tabla 5.1 se muestra un cuadro resumen de las medidas de tiempos correspondientes a las operaciones que no son de captura. El tiempo de ejecución y el tiempo medido coinciden en los casos en que no hay que esperar mensaje de respuesta. En las otras situaciones, el tiempo medido es bastante mayor que el de CPU debido a los bloqueos esperando mensajes.

Por ejemplo, en la inicialización se obtienen tiempos elevados debido a que se lleva a cabo una secuencia de varias operaciones en que intervienen negociaciones del DHCP, apertura de canales, etc. que a su vez se bloquean en espera de ciertos mensajes. En el caso de operaciones como cambiar el formato, además de estar también formadas por varios procesos, hay que tener en cuenta que una vez comienza el procedimiento la cámara sigue inundando el buffer de porciones de imagen hasta que se consigue parar la adquisición. Todos estos paquetes de imagen que no interesan en el procedimiento deben ser analizados y descartados en busca de las pertinentes confirmaciones.

**Tabla 5.1: Tiempos de las operaciones de no captura.**

| Test | Tcpu peor caso (ms) | Tcpu medio (ms) | T medio (ms) | Número muestras |
|---|---|---|---|---|
| Inicializar | 176.16 | 141.89 | 3091.93 | 2 |
| Keep_Streaming | 10.16 | 9.82 | 9.82 | 240 |
| Parar adquisición | 20.08 | 20.08 | 220.09 | 1 |
| Desconectar cámara | 29.54 | 29.54 | 429.54 | 1 |
| Reset | 10.27 | 10.27 | 10.27 | 1 |
| Cambiar formato | 98.96 | 91.80 | 1891.84 | 5 |
| Cambiar velocidad | 172.17 | 172.17 | 1695.08 | 1 |

En la Tabla 5.2 se muestran los tiempos de CPU que conlleva realizar las capturas en distintos formatos de imagen. Este es el tiempo que el driver desarrollado emplea en procesar cada imagen sin tener en cuenta bloqueos. En función de estos datos, se puede calcular la tasa media máxima (empleando el Tcpu medio) a la que podríamos pedir datos de imagen a la cámara (los datos de cabeceras van aparte). Las tasas son relativamente similares entre los diferentes formatos. Esto es lógico porque la aplicación emplea un tiempo proporcional en procesar los datos (a más datos, más tiempo). Sin embargo, esta tasa tiende a ser mejor (mayor) cuando crece el tamaño de la imagen porque con porciones pequeñas existe más overhead por las cabeceras.

El driver para cámara analógica que ya existía en MaRTE permitía tasas de 25 fps a resolución RGB 8 Packed 640x480. Con los datos manejados en el driver de este proyecto, nos aproximamos al valor obtenido anteriormente con la captura analógica. Ahora se consiguen unos 20 fps donde anteriormente se alcanzaban 25 fps. Sin embargo, el potencial de la cámara digital viene al poder manejar tasas mucho mayores cuando se cambia el formato de la imagen. Esta característica no existía anteriormente: todas las capturas eran con los mismos parámetros.

**Tabla 5.2: Tiempos de las operaciones de captura. La primera muestra de las 200 realizadas ha sido desechada por posibles fallos de cache.**

| Test | Tcpu peor caso (ms) | T cpu medio (ms) | Tasa media potencial de captura (MBps) | Tasa media potencial de captura (fps) |
|---|---|---|---|---|
| Mono 8 100x100 | 0.78 | 0.68 | 14.64 | 1464.95 |
| RGB 8 Packed 100 x 100 | 2.23 | 1.88 | 15.92 | 530.93 |
| Mono 8 320x240 | 5.51 | 4.50 | 17.05 | 222.04 |
| RGB 8 Packed 320x240 | 22.90 | 12.86 | 17.05 | 77.76 |
| Mono 8 640x480 | 17.67 | 16.99 | 18.07 | 58.84 |
| RGB 8 Packed 640x480 | 50.63 | 50.59 | 18.21 | 19.76 |

Lamentablemente, existen otras limitaciones que alejan la tasa de bits real del límite teórico calculado.

Por un lado, la red Ethernet empleada. Si bien en la especificación de la cámara se afirma que a máxima resolución la tasa es de 80 fps, este es un límite fijado para Gigabit Ethernet. Sin embargo, en el caso manejado, MaRTE no disponía de drivers para tarjetas a 1 Gbps por lo que la infraestructura tiene un límite de 100 Mbps (12.5 MBps) de los cuales se destina un margen para cabeceras. Además, no es aconsejable exigir el 100% de rendimiento al enlace de datos porque las probabilidades de error aumentan.

El otro problema que se ha podido ver tiene su origen en el driver Ethernet de MaRTE. Sólo ha permitido manejar tasas de transmisión sostenidas de 44 Mbps (5.5 MBps). Superando este límite, el driver empieza a informar de desbordamientos de buffer y errores de recepción que disparan el tiempo de captura de una imagen. Este parece ser un comportamiento erróneo del driver de bajo nivel que aparenta no recuperarse bien de las situaciones de desbordamiento. En realidad, no debería suponer problema alguno ya que si el buffer se desbordara, la consecuencia sería la pérdida de paquetes, en ningún caso error.

Lo mismo ocurre con el controlador GigE Vision desarrollado: cuando se pierden paquetes de una imagen esto no implica un error. En un sistema basado en UDP en el que el intercambio de paquetes es no fiable, la pérdida debe asumirse de este modo. Lo único que se comprueba cada vez que llega una cabecera de imagen es que los paquetes siguientes pertenecen a la misma imagen, pero que si llega una cabecera nueva, se acepta la nueva imagen. De esta forma, la cámara puede enviar varias imágenes hasta que se consiga una captura completa y es por ello por lo que aumenta el tiempo.

Por lo tanto, dadas las limitaciones del driver y de la conexión Ethernet, las pruebas han revelado que la tasa máxima de transmisión de datos es de 5.5 MBps. Los resultados obtenidos se muestran en la Tabla 5.3. Las capturas de formatos de 100x100 se han hecho a menor velocidad (2 MBps) porque al ser pocos paquetes por imagen, el driver Ethernet daba problemas a la hora de entregarlos.

No obstante, a pesar de los problemas detectados, aún se cuenta con capacidad de mejora respecto al driver analógico en resoluciones por debajo de 320x240. La mejora es en estos casos muy notable. Se pasa de la tasa analógica de 25 fps en todos los casos a aproximadamente 65

fps en blanco y negro 320x240 y color 100x100. Incluso se consiguen tasas aún mejores con regiones de 100x100 en blanco y negro: 199 fps frente a 25 fps.

Sin duda, el principal reto en un futuro deberá ser mejorar el driver Ethernet para que el controlador desarrollado pueda alcanzar su potencial.

**Tabla 5.3: Resultados tras 200 muestras.**

| Test | T medio (ms) | Tasa media real de captura (MBps) | Tasa media real de captura (fps) |
|---|---|---|---|
| Mono 8 100x100 | 5.03 | 1.98 | 198.81 |
| RGB 8 Packed 100 x 100 | 15.16 | 1.98 | 65.96 |
| Mono 8 320x240 | 15.42 | 4.98 | 64.81 |
| RGB 8 Packed 320x240 | 43.79 | 5.26 | 22.83 |
| Mono 8 640x480 | 58.66 | 5.24 | 17.04 |
| RGB 8 Packed 640x480 | 178.03 | 5.18 | 5.62 |

# 6. Conclusiones y trabajo futuro

## 6.1. Conclusiones

Se ha satisfecho el principal objetivo del proyecto:

- El **desarrollo de un driver** para cámaras digitales Ethernet basadas en el estándar GigE Vision en el sistema operativo de tiempo real MaRTE.

Para la consecución de este objetivo principal, ha sido necesario cubrir objetivos secundarios adicionales que han producido unos resultados para MaRTE muy destacables:

- **Pila Ethernet, IP y UDP** independiente y utilizable en otras aplicaciones.
- **Servidor DHCP** mínimo.

Durante el desarrollo del controlador GigE Vision se han seguido las siguientes directivas:

- Se ha creado una interfaz de usuario que permite un **uso lo más sencillo posible**.
- El desarrollo ha tenido en todo momento en cuenta al **estándar**. Esto permite utilizar el driver con cualquier cámara que lo cumpla. Sin embargo, es posible que algunos pequeños cambios en aspectos como direcciones de registros deban ser realizados.
- Se ha diseñado de forma **modular y ampliable**. Además de la propia implementación del estándar se han obtenido dos resultados independientes que pueden ser empleados en otros desarrollos: la pila Ethernet, IP y UDP y el servidor DHCP.
- **Aplicable a los entornos propuestos**. Además, gracias a las comprobaciones que se realizan sobre puertos UDP y direcciones IP, es posible utilizar el proyecto en una red con Ethernet dedicada o compartida.

Sin embargo, el controlador desarrollado no es todo lo eficiente que cabría esperar de los resultados teóricos. Las limitaciones del driver Ethernet ya existente han supuesto una merma de los resultados esperados respecto al referente establecido con la antigua cámara analógica. Aún así, el empleo de la característica ROI ha permitido capturar un número de imágenes por segundo mucho mayor que con la cámara anterior. Es común que las aplicaciones de control en tiempo real (las que probablemente empleen este driver) necesiten únicamente zonas limitadas de la imagen. Por ello, dado que los tiempos para ROIs medianas y pequeñas son bastante buenos, el controlador podrá ser empleado satisfactoriamente.

## 6.2. Trabajo futuro

A pesar de que los objetivos han sido cubiertos, siempre queda espacio para **mejoras y ampliaciones** que van surgiendo a medida que se progresa en el trabajo realizado. Algunas de las ideas son:

- La principal mejora viene de la mano del **driver Ethernet**. Desde el punto de vista del controlador creado, se ha alcanzado un límite cuya responsabilidad de mejora corresponde a otros componentes del sistema operativo MaRTE. Es necesario que el controlador Ethernet acepte velocidades de transmisión mayores, probablemente creándolo nuevamente para tarjetas de red Gigabit Ethermet.
- **Ampliar el controlador con características adicionales del estándar.** Por ejemplo, implementar el canal de mensajes para que la cámara informe de eventos que le acontecen. Otro ejemplo sería dar la posibilidad de controlar varias cámaras creando más canales de control.
- **Mejorar el servidor DHCP.** Actualmente el servidor responde con mensajes que tienen siempre los mismos parámetros y que son válidos para configurar los aspectos de direccionamiento básicos. Sin embargo, si se quiere emplear con otros clientes, puede que estos pidan otros parámetros de configuración. Dotando al servidor de más inteligencia se podría responder a peticiones de distintos tipos. Por otro lado, ahora el servidor no se queda a la escucha de peticiones mas que cuando la cámara va a solicitar por primera vez sus parámetros de configuración. Sería aconsejable que el servidor estuviera en funcionamiento de forma constante y que contestara a otros dispositivos y sobretodo a las peticiones que la cámara vuelve a enviar cuando le caduca la concesión de sus parámetros.
- Ampliar el campo de uso a **más arquitecturas de red**, por ejemplo adaptarlo para ser empleado desde otras redes Ethernet. Actualmente todos los componentes se encuentran en la misma red local (servidor DHCP y aplicación y cámara GigE Vision). Se podría probar a posicionar estos componentes en redes distintas y configurarlos de forma que puedan utilizarse como hasta ahora.
- Añadir **soporte a más formatos de imagen**.
- Aunque el estándar no lo deja demasiado claro, parece que lo más recomendable es **emplear el fichero XML que describe características de la cámara** para saber qué registros permiten modificar formatos, velocidades, etc. En este proyecto se optó por pedir al fabricante de la cámara la especificación de los registros, los cuales son válidos para cualquier cámara de Allied Vision Technologies pero no lo serán en el caso de cámaras de otros fabricantes. Para evitar hacer de forma manual el cambio de direcciones (sólo es necesario modificar algo más de una docena de constantes) lo ideal sería que la aplicación manejara este fichero XML para adaptar de forma automática el driver a cualquier tipo de dispositivo.

# Bibliografía

BARNES, J. 2006. Programming in Ada 2005. Harlow (Inglaterra): Pearson Education. ISBN: 978-0-321-34078-8

ESCALERA, A. 2001. Visión por computador: fundamentos y métodos. Madrid: Pearson Educación. ISBN 84-205-3098-0.

FLORIST. 2011. Florist - Implementación de las POSIX Ada bindings [sitio web]. [Consulta: 4 diciembre 2011]. Disponible en: http://www.cs.fsu.edu/~baker/florist.html

GIGE VISION STANDARD. 2010. GigE Vision, Video Streaming and Device Control over Ethernet Standard. Ann Arbour (EE.UU): Automated Imaging Association.

GONZÁLEZ, M.; GUTIÉRREZ, J. J.; PÉREZ, H. 2009. Apuntes de Programación en Lenguaje Ada (2010). En: Open Course Ware (OCW) Universidad de Cantabria [en línea]. Santander: Universidad de Cantabria, parte 1, cap. 2 [Consulta: 13 noviembre 2011] Versión pdf. Disponible en: http://ocw.unican.es/ensenanzas-tecnicas/programacion-en-lenguaje-ada/material-de-clase-2/cap2-basico.pdf

GONZÁLEZ, M.; PALENCIA, J. C. 2009. Apuntes de Sistemas de Tiempo Real. Basado en tutorial realizado por Software Engineering Institute, Canergie Mellon University (EE.UU.). Santander: Universidad de Cantabria, cap. 1.

MACHINE VISION ONLINE. 2011. GigE Vision Standard [sitio web] [Consulta: 13 noviembre 2011]. Disponible en: http://www.machinevisiononline.org/vision-standards-details.cfm?type=5

MARTE OS. 2011. MaRTE OS [sitio web]. Santander: Universidad de Cantabria. [Consulta: 13 noviembre 2011]. Disponible en http://marte.unican.es/

PACKETH. 2011. PackETH - Ethernet packet generator [sitio web]. [Consulta: 4 diciembre 2011]. Disponible en: http://packeth.sourceforge.net/

RFC1533. 1993. DHCP Options and BOOTP Vendor Extensions. Lewisburg (EE.UU.): Bucknell University [en línea] [Consulta: 13 noviembre 2011]. Disponible en: http://tools.ietf.org/html/rfc1533

RFC2131. 1997. Dynamic Host Configuration Protocol. Lewisburg (EE.UU.): Bucknell University [en línea] [Consulta: 13 noviembre 2011]. Disponible en: http://tools.ietf.org/html/rfc2131

RFC SOURCE BOOK. 2011. RFC Source Book [sitio web]. San Diego (EE.UU.): Network Sorcery, Inc. [Consulta: 13 noviembre 2011]. Disponible en: http://www.networksorcery.com/enp/protocol/bootp/options.htm

STALLINGS, W. 2004. Comunicaciones y Redes de Computadores. Madrid: Pearson Educación. ISBN 84-205-4110-9.

VALLEJO, E. 2008. Apuntes de Introducción a las Redes de Computadores. Santander: Universidad de Cantabria, cap. 8.

WIRESHARK. 2011. Wireshark [sitio web]. [Consulta: 4 diciembre 2011]. Disponible en: http://www.wireshark.org/

# Glosario

**ARP -** Address Resolution Protocol (*Protocolo de Resolución de Direcciones*) es un protocolo de nivel de red que permite encontrar la dirección hardware (Ethernet MAC) que corresponde a una diracción IP.

**CRC** - Cyclic Redundancy Check (*Comprobación de Redundancia Cíclica*) es una función que recibe un flujo de datos de cualquier longitud y devuelve un valor de longitud fija. El término es empleado tanto para referirse a la función como para el propio resultado. Las CRC son populares porque permiten detectar de forma muy efectiva errores ocasionados por ruido en los canales de transmisión. Las CRC se calculan tanto en el origen como en el destino. Sus valores se comparan para detectar errores.

**GenICam** - Estándar que define una interfaz de programación genérica para todo tipo de cámaras. Independientemente de la tecnología que subyace en cada una de ellas, la API es siempre la misma. Se emplea para no tener que utilizar distintos controladores para cámaras distintas.

**GNAT -** GNAT es un compilador para el lenguaje de programación Ada. Ha sido escrito casi por completo en el propio lenguaje Ada e implementa todos los anexos del estándar, habiendo sido certificado para ello.

**GPS** - GNAT Programming Studio. Entorno de desarrollo para Ada que emplea el compilador GNAT.

**Inteligencia artificial -** Disciplina que se encarga de construir procesos ejecutados sobre arquitecturas físicas no vivas que en función del entorno percibido producen acciones o resultados que maximizan una medida de rendimiento determinada.

**Modo broadcast** - También llamado modo difusión, es un modo de transmisión de datos en que estos son recibidos en múltiples receptores de forma simultánea.

**Modo unicast -** Modo de transmisión de datos en que estos viajan desde un único emisor a un único receptor.

**XML** - XML (*eXtensible Markup Language*) es un metalenguaje extensible que permite definir la gramática de lenguajes específicos.